مفاهيم نظم التشغيل

Operating system Concepts



عبدالرحمن أحمد محمد عثمان

دكتوراة في علوم الحاسب من جامعة الخرطوم

الطبعة الثانية

مارس 2013

طبعة الكترونية مجانية

للتواصل والمقترحات والتصويبات: operatingsystem13@gmail.com

مقدمة

الحمد لله الذي علم بالقلم علم الإنسان ما لم يعلم، والصلاة والسلام على الرسول الأكرم وعلى آله وصحبه وسلم، وبعد

لقد أصبح الحاسب مستخدما في كل مجال وفي كل مكان، فهو يتحكم في السيارة وفي الطائرة، وهو في الجيب، وهو في المطبخ وهو بالمصنع وهو بالصرافات الآلية يتحكم في أموالنا، وهو في الأرصاد الجوي يتنبأ بالأحوال الجوية، وهو في الإذاعة والتلفزيون يعرض البرامج والأخبار وهو في المكتب يدير أعمالنا وهو في البيت يساهم في الترفيه والاتصالات ..الخ..

لا غنى لنا عن الحاسب، ولا غنى للحاسب عن نظام التشغيل. فهو قلب الحاسب ومحركه و بدونه لا يمكن التعامل مع الحاسب ولا تشغيله ولا استخدامه. لذلك معرفة نظام التشغيل وفهم طريقة عمله من الأساسيات التي يبنى عليها الكثير من علوم الحاسب، فكل دارس يريد سبر أغوار الحاسب والتمكن منه لابد له من معرفة نظام التشغيل معرفة متعمقة ومتفحصة توسع إدراكه وتمكنه من التعامل مع الحاسب بالصورة المثلى.

جاء هذا الكتاب ليخدم هذا الغرض، فهو يشرح مفاهيم نظم التشغيل وطريقة عمله، وقد قسم الكتاب إلى أثني عشر بابا هي كما يلي:

مفهوم نظام التشغيل في الباب الأول، الحاسب وبنية نظام التشغيل في الباب الثاني، العمليات كانت بالباب الثالث، الجدولة وكيفية تقسيم زمن المعالج بين العمليات أفردنا لها الباب الرابع، بينما تحدثنا عن الخيط وبرمجته في الباب الخامس، وبالباب السادس تحدثنا عن تزامن العمليات وكيفية تعاون عدد من العمليات لإنجاز عمل واحد، اختناق العمليات على الموارد وكيفية الوقاية منه ومعالجته بالباب السابع، أما كيف يدير نظام التشغيل الذاكرة الرئسية فقد أفردنا لها الباب الثامن، وفي الباب التاسع كان الموضوع كيف يستخدم نظام التشغيل القرص الصلب كإمتداد للذاكرة الرئيسية فيما يسمى الذاكرة الظاهرية. أجهزة الدخل والخرج التي تتحكم في دخول وخروج المعلومات من وإلى الحاسب وكيف يتحكم نظام التشغيل فيها كانت في الباب العاشر. طرق تخزين البيانات وحفظها في شكل ملفات وكيف يقوم نظام التشغيل بذلك كانت

بالباب الحادي عشر. في الباب الثاني عشر تحدثنا عن النظم الموزعة وكيف تدير بعض نظام التشغيل مثل هذه النظم.

في الختام نقول أن هذا الكتاب هو جهد المقل، فإن كانت فيه أخطاء فمن أنفسنا ومن الشيطان وإن وفقنا فيه فمن رب العالمين، نسأل الله أن يقينا به حر جهنم وينفع به طلابنا في الجامعات العربية.

أخيرا، لا يخلو عمل من أخطاء والكمال لله وحده، لذلك نرجو من القارئ الكريم مساعدتنا بالتصويبات والإقتراحات التي تفيدنا في الطبعة القادمة بإذن الله.

نرجو من القارئ الكريم التواصل معنا عبر الايميل

operatingsystem13@gmail.com

لاضافة مقترحات أو تصويب اخطاء.

المؤلف

جدول المحتويات

الباب الأول:

مدخل

الباب الأول

مدخل

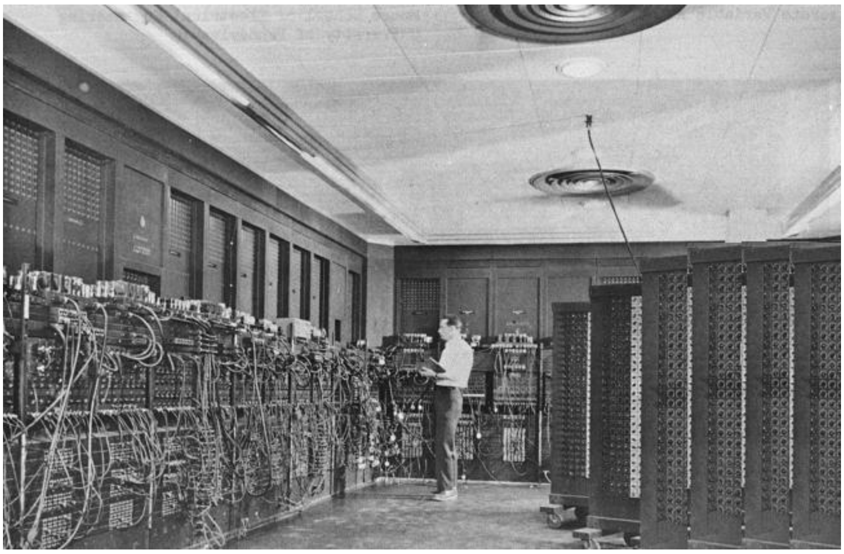
الحاسب جهاز يتكون من مكونات مادية (hardware) ومكونات برمجية (software). المكونات المادية تمثل جسد الحاسب، أي الأجهزة الملموسة من شاشة، ولوحة المفاتيح ، ومعالج، وذاكرة، وغيرها. أما المكونات البرمجية فتمثل روح الحاسب (هي التي تتحكم في المكونات المادية وتوجه عملها). وليس من السهل استخدام المكونات المادية للحاسب بدون برامج. فالحاسب بدون برامج كالجسد بلا روح أو كالسيارة بلا وقود. أهم جزء في هذه البرامج والذي تعتمد عليه البقية في عملها هو نظام التشغيل.

1.1. حاسبات بدون نظم تشغيل

بدأت الحاسبات قديما بلا برامج وبلا نظم تشغيل، وكان العمل كله يتم بلغة الآلة (شفرة مكونة من أصفار و وحائد)، وبالتالي لا يتعامل مع الحاسب إلا المهندسين المختصين. حتى المبرمج العادي كان يسلم برامجه للمختص ليدخلها له في الحاسب وينفذها له، ثم يسلمه النتائج .

إذن فللتعامل مع الحاسب لابد من شخص مختص يسمى المشغِّل يجلس بقرب الحاسب ليساعد المبرمجين والمستخدمين على التعامل مع الحاسب. يقوم المشغِّل باستلام برامج المبرمجين وبيانات المدخلين حيث يقوم بإدخالها وتشغيلها ثم يسلمهم النتائج مطبوعة على ورق (قد يكون هنالك طابور من المبرمجين الذين يريدون استخدام الحاسب).

كان الحاسب في ذاك الوقت يستخدم البطاقة المثقبة للإدخال والطابعة للإخراج، فلا لوحة مفاتيح ولا شاشة ولا غيرها.



الشكل (1-1): حاسب بحجم الغرفة والمشغل بداخله

كانت الأجهزة في ذاك الوقت :

- ضخمة، الشكل (1-1).
- بطيئة ومحدودة الأغراض.
- باهظة السعر ولا يستطيع امتلاكها أشخاص عاديين.

وكانت البرامج :

- تكتب بلغة الآلة.
- تدخل بالبطاقة المثقبة.
- تقوم بعمليات حسابية ورياضية محدودة.
- تظهر مخرجاتها على الورق عبر الطابعة.

لذلك كان التعامل مع الحاسب أمراً عسيراً و شاقاً لا يستطيعه إلا المختصين المهرة.

1.2. بداية نظم التشغيل

بدأ العمل بتصميم برامج تؤدي جزء من مهام المشغل وبدأت أعباؤه تقل تدريجيا، إلى أن تم إحلال كامل للمشغِّل ببرامج تقوم بكل مهامه السابقة، فكانت نظم التشغيل التي وفرت الكثير من الوقت المستغرق للتعامل مع الحاسب ليستفاد منه في تطوير نظام التشغيل والتطبيقات الأخرى.

ثم تطورت نظم التشغيل من نظم تكتب أوامرها في شكل نصوص مثل DOS إلى نظم تشغيل رسومية في شكل أيقونات ورموز يستطيع كل شخص التعامل معها، وبالتالي أصبح الكل يتعامل مع الحاسب بكل سهولة ويسر.

بروز نظم التشغيل جاء نتيجة لجهد كبير بذله المطورون لهذه النظم. فكل ما زاد الجهد المبذول في تطوير هذه النظم كانت النتيجة نظم سهلة الاستخدام وملائمة لمتطلبات المستخدمين.

الآن وبعد التطور الكبير في صناعة المكونات المادية والبرامج أصبحت أجهزة الحاسب تحمل على الكف ونظم التشغيل والبرامج تستخدم بنقرة من الفأرة أوضغطة على لوحة المفاتيح أو كلمة على المايكروفون، (شكل 1-2).



شكل (1-2): حاسب كفي بنظام تشغيل ويندوز موبايل.

1.3. تعريف نظام التشغيل

نظام التشغيل هو ذلك البرنامج الذي نراه عندما نفتح الحاسب ولا يفارقنا إلا عند إغلاقه. وهو أول برنامج يثبت على الحاسب ليدير جميع موارده ويتيح للمستخدم واجهة مستخدم (user interface) تمكنه من التعامل مع المكونات المادية بكل سهولة ويسر.

1.3.1. أهداف نظام التشغيل الرئيسية هي:

- تنفيذ تطبيقات المستخدم.
- توفير بيئة مناسبة وملائمة للاستخدام (convenient).
- الاستفادة القصوى من الموارد وذلك بجعلها تعمل بشكل فعال (efficient).

1.3.2. شرح تعريف نظام التشغيل

الموارد (resources):

موارد الحاسب تشمل المكونات المادية من لوحة المفاتيح وشاشة وطابعة، وغيرها، وكذلك الملفات والبرامج وصفحات الويب وما شابه.

الواجهة (user interface):

يتعامل المستخدم مع البرامج التطبيقية وموارد الحاسب من خلال واجهة استخدام (user interface). فمعظم نظم التشغيل اليوم توفر واجهة مستخدم رسومية ((graphical user interface (GUI))، حيث تمثل الأيقونات المزايا المتوفرة بالنظام.

تنفيذ برامج المستخدم:

يقوم نظام التشغيل بتحميل برامج المستخدم في الذاكرة وتشغيلها بالمعالج، وتوفر معظم نظم التشغيل الحديثة تحميل وتشغيل اكثر من برنامج في وقت واحد (تعدد البرامج).

المقصود بإدارة الموارد هو:

- حجز المورد (allocate) للبرنامج الذي يطلبه، ثم تحريره (free) بعد الإنتهاء منه وإتاحته لتستفيد منه برامج أخرى.
- استخدام المورد بكفاءة والاستفادة منه الاستفادة القصوى: مثلا إذا كان المعالج ينفذ في برنامج معين، وأحتاج هذا البرنامج إلى معلومة من لوحة المفاتيح (قد يستغرق وصول المعلومة وقتا ليس بالقصير مقارنة مع سرعة المعالج)، في هذه الحالة سيقوم نظام التشغيل بالاستفادة من المعالج في تنفيذ برنامج آخر ريثما تصل المعلومة من لوحة المفاتيح، هنا يكون نظام التشغيل قد استفاد من زمن المعالج في هذه الفترة.

- العدل في استخدام الموارد: يمنع نظام التشغيل البرامج من حجز الموارد واستخدامها لمدة طويلة.

1.3.3. نظام التشغيل كبرنامج تحكمي:

يتحكم نظام التشغيل في تشغيل البرامج الأخرى ويدير عملية تنفيذها لتفادي الأخطاء وتجنب الاستخدام الغير مرشد لموارد الحاسب وحماية البرامج عن بعضها البعض وعن نظام التشغيل.

1.4. ما هو نظام التشغيل ؟

إذا قمت بتثبيت نظام التشغيل ويندوز ستجد معه الكثير من البرامج مثل متصفح الإنترنت، ومشغل الوسائط (media player)، والرسام والمفكرة والكثير من الألعاب وغيرها من البرامج، هل نعتبر هذه البرامج جزء من نظام التشغيل؟ للمستخدم العادي نقول نعم !. أما علميا فنقول:

أن نظام التشغيل هو ذلك الجزء الذي يتعامل ويدير المكونات المادية مباشرة، وهو النواة (kernel) التي لا نراها و لا نتعامل معها مباشرة، لكننا لا نستغني عن خدماتها التي هي سبب تشغيل بقية البرامج والواجهات التي نتعامل معها.

البعض يضيف إلى نظام التشغيل الغلاف والواجهة الرسومية التي من خلالها نستخدم النظام.

1.5. مكونات نظام الحاسب

نظام الحاسب (أو الحاسب) هو عبارة عن مكونات المادية و مكونات برمجية، يمكن تفصيل هذه المكونات بصورة أدق إلى الآتي:

- مكونات الحاسب المادية (computer hardware).
- النواة (kernel).
- واجهات نظام التشغيل (operating system interfaces).
- تطبيقات المستخدم (applications).
- المستخدم (user).

1.5.1. المكونات المادية

يتكون الحاسب من معالج أو أكثر، ذاكرة رئيسية، أجهزة تخزين مثل القرص الصلب، أجهزة دخل وخرج، نواقل لتوصيل هذه الأجهزة مع بعضها.

1.5.2. النواة

تدير النواة مكونات الحاسب المادية. وتنقسم إلى خمسة أجزاء رئيسية هي:

- جزء مسئول عن إدارة المعالج يسمى مدير العملية.
- جزء مسئول عن الذاكرة الرئيسية يسمى مدير الذاكرة.
- جزء مسئول عن إدارة أجهزة الدخل والخرج يسمى مدير الأجهزة.
- جزء مسئول عن أجهزة التخزين ويسمى مدير الملفات.
- جزء مسئول عن التعامل مع الشبكة يسمى مدير الشبكة.

سيكون هنالك باب مفصل عن عمل كل جزء من هذه الأجزاء.

1.5.3. وجهات نظام التشغيل

توفر واجهات نظام التشغيل للمستخدم وتطبيقاته الإتصال مع النواة. وهنالك ثلاث أنواع من واجهات نظام التشغيل هي:

- واجهة المستخدم الرسومية (GUI).
- الغلاف (shell) أو مترجم الأوامر (command line interpreter).
- واجهة نداء النظام (system call interface).

1.5.3.1. واجهة المستخدم الرسومية

تعتبر أعلى مستوى حيث نتعامل معها مباشرة عبر الإيقونات والقوائم والنوافذ التي نشاهدها على سطح المكتب. هذه الواجهة تسمح للمستخدم بالتعامل مع نظام

التشغيل بطريقة سهلة وملائمة، فمثلا يستطيع المستخدم طلب أمر بنقرة ماوس. من أمثلة واجهات المستخدم سطح المكتب في ويندوز، و X-Window في لينكس.

في هذا المستوى لا يعلم المستخدم ولا يهتم بتفاصيل النواة. لا يعتبر هذا المستوى جزء من نظام التشغيل بل هو مكون برمجي أضيف ليمكن المستخدم من التعامل مع نظام التشغيل.

1.5.3.2. الغلاف أو مترجم الأوامر

يسمح للمستخدم (الخبير) بالتعامل مع النواة مباشرة من خلال كتابة أوامر نصية، وهو يعتبر في نفس مستوى واجهة المستخدم.

في المستوى الثالث تستخدم التطبيقات واجهة نداء النظام لطلب الخدمات التي يوفرها نظام التشغيل.

1.5.3.3. نداء النظام System call

إذا احتاجت برامج المستخدم خدمة معينة من نظام التشغيل تستخدم ما يسمى نداء النظام (system call). ذلك لأن برامج المستخدم غير مسموح لها بالوصول المباشر للمكونات المادية، وإنما نواة نظام التشغيل هي التي تستطيع فعل ذلك. بهذه الطريقة نضمن سلامة المكونات المادية وحمايتها من البرامج التطفلية ومن الاستخدام الخاطئ لها. ولكن احيانا تحتاج بعض تطبيقات المستخدم التعامل مع المكونات المادية، ولأن هذه التطبيقات لا تستطيع الوصول للمكونات المادية مباشرة، ستقوم بإرسال طلب إلى نظام التشغيل ليمدها بالمعلومات التي تريد من المكون المادي المعين.

التعامل مع المكونات المادية يوفرها نظام التشغيل في شكل خدمات، حيث يتم الطلب في شكل نداء النظام المناسب. حيث يوجد لكل خدمة نداء نظام خاص بها.

يتم تنفيذ نداء النظام في وضع النواة (kernel mode). ولكل استدعاء نظام رقم مرتبط به. يرسل هذا الرقم إلى النواة ليعرف نظام التشغيل ما هو استدعاء النظام المطلوب. عندما يرسل المستخدم هذا الرقم فهو في الحقيقية يستدعى روتين مكتبة

(library routine)، فيقوم الروتين بإرسال مقاطعة (issues a trap) لنظام التشغيل، ثم يمرر رقم الاستدعاء ومعطياته إلى النواة (باستخدام مسجلات معينة). تقوم النواة بتنفيذ الروتين وترسل النتائج للمستخدم عبر مسجل معين. إذا كانت النتائج كبيرة الحجم (لا يستطيع المسجل تخزينها)، سترسل بطريقة أخرى مثل استدعاء الروتين copy_to_user لتخزينها في موقع ما بالذاكرة.

من نداءات النظام المشهورة في نظم التشغيل لينكس:

open, read, write, close, wait, exec, fork, exit, kill

معظم نظم التشغيل اليوم تحتوي على كم هائل من نداءات النظام. مثلا يوجد في لينكس حوالي 319 نداء نظام، وفي FreeBSD توجد حوالي 330 نداء نظام.

كم استدعاء نظام يوجد في ويندوز XP؟ وهل يختلف عن عدد نداءات النظام الموجودة في نسخ ويندوز الأخرى (مثل Vista و Windows 2000)؟ أذكر نداءات النظام المشهورة في ويندوز ؟



شكل رقم (3-1): مكونات نظام الحاسب

1.6. دعم تعددية البرامج

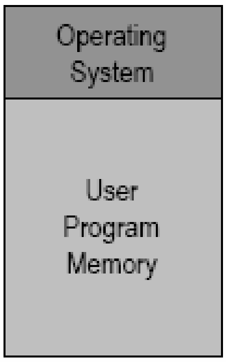
نظم التشغيل يوفر بيئة لتنفيذ البرامج، ويركز نظام التشغيل على فعل ذلك بكفاءة تزيد من الاستفادة من موارد الحاسب. قديما كان نظام التشغيل يسمح لبرنامج واحد بالعمل وهذا يسمى أحادية البرامج، أما الآن فيدعم نظام التشغيل تنفيذ أكثر من برنامج في وقت واحد.

1.6.1. الأنظمة أحادية المهام

كان دور نظام التشغيل هو تحميل برنامج واحد وتنفيذه، ثم بعد إكتماله، يتم تحميل برنامج آخر وتنفيذه، وهكذا.

يعتبر نظام التشغيل غير كفؤ لأنه يحمل برنامج واحد كل مرة، ويظل هذا البرنامج يعمل إلى أن ينتهي، خلال عمل هذا البرنامج قد تكون هنالك موارد متاحة لكننا لانستفيد منها لأنه لا يوجد برنامج بالذاكرة سوى هذا البرنامج. أيضا يستغرق تحميل البرنامج وقتا ليس بالقصير، مما يؤثر على أداء الحاسب.

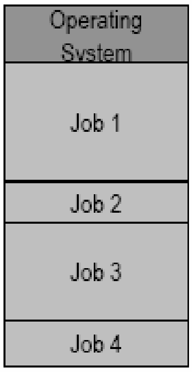
يسمى هذا النوع من نظم التشغيل أحادي المهام، حيث يوجد برنامج واحد بالذاكرة بالإضافة إلى نظام التشغيل، شكل (1-5).



شكل (1-4): شكل الذاكرة في نظام التشغيل أحادي البرنامج

1.6.2 تعدد البرامج (multiprogramming)

هنا يتعامل نظام التشغيل مع عدد من البرامج المخزنة بالقرص، حيث يحمل جزء من هذه البرامج بالذاكرة (شكل 1-5). يبدأ نظام التشغيل بتنفيذ أحد هذه البرامج في المعالج. إذا توقف هذا البرنامج عن التنفيذ لأي سبب، يقوم نظام التشغيل بتشغيل برنامج آخر في المعالج. بهذه الطريقة نكون قد استفدنا من زمن المعالج وجعلناه مشغولا معظم الوقت. فالغرض من تعدد البرمجة وجود أكثر من برنامج بالذاكرة بحيث يجد المعالج دائما برنامج جاهز للتنفيذ.



شكل (1-5): شكل الذاكرة في نظام التشغيل متعدد البرامج

1.6.3. المشاركة الزمنية (Time-sharing)

هي استمرار منطقي لتعدد البرامج. يقوم المعالج بخدمة العديد من المهام وذلك بإعطاء كل مهمة فترة زمنية قصيرة داخل المعالج، ويتنقل المعالج بين المهام بسرعة عالية جدا لدرجة أن كل مهمة تعمل وكأنها تستخدم المعالج لوحدها.

إذا احتاجت مهمة أن تنتظر عملية دخل أو خرج، يمكن الانتقال لمهمة أخرى مما يكسب النظام استغلال جيد لزمن المعالج.

1.6.4. مثال تشبيهي

إذا كنت تمتلك سيارة أجرة (المورد)، وأردت الاستفادة منها استفادة قصوي فلن تستطيع، لأنك ستنام وستأكل وستقوم بمهام أخرى غير العمل (قيادة السيارة). إذن

سيكون هنالك أوقات تكون فيها السيارة غير مستخدمة (idle)، هذا يشبه نظم التشغيل احادية المهام.

أما إذا كانت السيارة لثلاث أشخاص مثلا، فيمكنهم التناوب في قيادتها، وبالتالي يمكن أن تكون السيارة تعمل 24 ساعة باليوم، 7 أيام بالاسبوع. يكون هنالك شخص من الثلاث يقوم بقيادة السيارة ثم إذا ما أحتاج أن يرتاح أو أن يأكل مثلا يقوم شخص آخر من الثلاث بقيادتها (الشخص الجاهز، أي ليس لديه أي إلتزامات أخرى)، فهذا يشبه تعدد البرامج.

أما إذا كانت السيارة لثلاث أشخاص، وحددنا أن لكل شخص فترة زمنية معينة (مثلا لكل شخص ثلاث ساعات) يقود فيها السيارة، فسيقود الشخص الأول السيارة الثلاث ساعات الأولى ثم الثاني ثلاث ساعات أخرى ثم الثالث ثلاث ساعات، ثم مرة أخرى الأول ثلاث ساعات ، وهكذا، نقسم زمن استخدام السيارة بين مالكيها. هذا يشبة التقسيم الزمني في نظم التشغيل.

حيث تمثل السيارة المعالج أو موارد الحاسب والثلاث أشخاص يشبهوا البرامج التي تتشارك في استخدام هذه الموارد.

1.7. نظم التشغيل المعاصرة

هنالك الكثير من نظم التشغيل، تختلف باختلاف الأنظمة والأجهزة والأغراض، فمنها ما يستخدم لإدارة جهاز واحد شخصي ومنها ما يستخدم لإدارة أجهزة متعددة المعالجات ومنها ما يستخدم لإدارة أجهزة كفية ومنها يستخدم لإدارة أجهزة مضمنة ...الخ. من نظم التشغيل الشهيرة المستخدمة حاليا التالي:

- ميكروسوفت ويندوز (windows): نسبة عالية من الحاسبات تستخدم ويندوز والتي تعمل على معالجات إنتل والمعالجات المتوافقة معها. توجد منها عدة نسخ، مثل ويندوز XP وويندوز فيستا (تستخدم في الحواسيب الشخصية)، ومثل ويندوز advanced server وويندوز 2000 التي تستخدم في الخوادم (servers).
- يونيكس (UNIX): صمم في عام 1974 بواسطة Dennis Ritchie و Ken Thompson بينما كانا يعملان في معامل AT & T Bell. كان الهدف نظام

تشغيل صغير ومتنقل. ثم أنتشر في الجامعات مراكز البحوث في عام 1980. ومن نسخ ينكس المشهورة:

- V Unix
- BSD

وهو أول نظام تشغيل يكتب بالكامل بلغة برمجة عالية (high level language)، فهو مكتوب بلغة C.

- ماكنتوش (Mac): صمم ليعمل على أجهزة أبل ماكنتوش التي تنتشر في دور الطباعة والنشر، وهو قوي وسهل الاستخدام وقد أخذت ويندوز فكرة النوافذ وسطح المكتب من هذا النظام حيث كان أول نظم تشغيل يدعم الواجهات الرسومية والايقونات والقوائم على سطح مكتب.
- توزيعات لينكس (Linux): هي نسخة مصغرة من ينكس صممت لتعمل على الحاسبات الشخصية، وهو مفتوح المصدر حيث يتيح حرية تعديل الشفرة وإعادة التوزيع . ويوجد منها مئات التوزيعات أحدثها وأشهرها توزيعه أوبونتو التي تدعم كل لغات العالم واجهة وكتابة (بما فيها اللغة العربية).

1.8. أنواع أنظمة الحاسوب

نظام الحاسب هو عبارة عن مكونات مادية (hardwrae) وبرامج. تختلف أنظمة الحاسوب باختلاف الاغراض التي من أجلها صممت، وتتنوع في السرعة والحجم والشكل، سنتحدث عنها باختصار فيما يلي.

1.8.1. الحاسبات المركزية **Mainframe Systems**

الحاسبات المركزية عبارة عن جهاز حاسوب مركزي واحد تتصل به عدة طرفيات تسمى الطرفيات العمياء (dump terminal)، هذه الطرفيات عبارة عن شاشة ولوحة مفاتيح، ليس بها ذاكرة أو معالج وإنما تستخدم معالج وذاكرة الحاسب المركزي.

تركز أنظمة تشغيل هذا النوع من الحاسبات على التقسيم الزمني، فهنالك مستخدمين كثر متصلين بطرفيات يريدون الاستفادة القصوى من موارد حساب مركزي واحد، وبالتالي على نظام التشغيل إدارة الجهاز المركزي لخدمة هذا الكم من المستخدمين بحيث يكون زمن الاستجابة لكل مستخدم سريع (أقل من ثانية)، وحتى يشعر كل مستخدم أن الجهاز المركزي يخدمه لوحده.

1.8.2. الحاسبات الشخصية

هي حاسبات غالبا تكون بمعالج واحد وشاشة ولوحة مفاتيح وتخدم شخصا واحدا.

تركز انظمة تشغيل هذا النواع من الحاسبات على خدمة مستخدم واحد ودعم تعدد البرامج بحيث يستطيع المستخدم تشغيل أكثر من برنامج في وقت واحد. أيضا يهدف هذا النوع من نظم التشغيل على توفير بيئة ملائمة للمستخدم واستجابة سريعة لطلبات المستخدم.

أمثلة لهذا النوع نظام التشغيل ويندوز XP، ويندوز فيستا، ماكنتوش، لينكس، و FreeBSD. هذا النوع يسمى مستخدم واحد متعدد المهام (Single user Multi-task)

أما نظم تشغيل القديمة للحاسبات الشخصية مثل دوس (Disk Operating System (DOS)). لا تدعم تعدد البرمجة، فهي لا تنفيذ أكثر من برنامج في نفس الوقت، ونطلق عليها مستخدم واحد أحادي المهام (Single user Single task)).

1.8.3. الأجهزة متعددة المعالجات (**Multiprocessor**)

بعض البرامج تحتاج سرعة عالية بحيث لا تكفي سرعة المعالج الواحد مهما بلغت، الحل هو استخدام أكثر من معالج لتنفيذ المهام وقد تتواجد عدة معالجات في صندوق واحد فيما يسمى تعدد المعالجات (multiprocessor).

يتميز تعدد المعالجات بأنه:

- قليل التكلفة مقارنة بالأنظمة الأخرى (حيث تتشارك المعالجات في بقية موارد الجهاز ، فالذاكرة مشتركة واللوحة الأم واحدة).
- سريع (لأن تكلفة الاتصال قليلة، فغالبا تستخدم المعالجات النواقل الداخلية في تبادل المعلومات).
- بسيط لأنه يستخدم ذاكرة مشتركة، فكل المعالجات تتشارك فيها.
- زيادة الإعتمادية: إمكانية الاستمرارية حتى ولو تعطلت بعض المعالجات.

أصبحت الحسابات ذات المعالجات متعددة النواة (multi-core)،مثل المعالج ثنائي النواة (dual core) والمعالج رباعي النواة (quad core)، منتشرة ومتوفرة وبعد قليل ستحل محل الحاسبات أحادية المعالج. تعتبر المعالجات متعددة النواة مثال مبسط لتعدد المعالجات.

السبب في ظهور تعدد المعالجات هو انخفاض سعر المعالجات والحاسبات الشخصية.

نظم التشغيل التي تدير هذا النوع من النظم تنقسم إلى نوعين هما:

1.8.3.1. المعالجات المتماثلة Symmetric multiprocessing (SMP)

هنا كل المعالجات متساوية وينفذ كل معالج مهمة منفصلة، وتتصل هذه المعالجات مع بعضها البعض عند الحاجة.

1.8.3.2. المعالجات غير المتماثلة Asymmetric multiprocessor

يوجد معالج رئيسي واحد يتحكم في النظام وينفذ مهمة رئيسية كبيرة، بينما بقية المعالجات تنفذ ما يأمرها به هذا المعالج الرئيسي، فهو قد يقسم المهمة الكبيرة إلى أجزاء صغيرة يوزعها على بقية المعالجات ثم يجمع النتائج بعد التنفيذ.

1.8.4. الأنظمة الموزعة

توزيع العمل عبر الشبكة إلى عدة حاسبات، هذه الحاسبات قد تكون قريبة أو بعيدة، وكل حاسب لديه معالجه، ذاكرته، وقرصه وأجهزته الطرفية الخاصة به. وقد تكون هذه الأجهزة متباينة في المكونات المادية ونظم التشغيل التي تديرها.

الميزات:

- التشارك في الموارد.
- زيادة سرعة التنفيذ.
- توزيع الحمل بين هذه الحاسبات (load balancing).

العيوب:

- زمن الاتصال بين أجزاء النظام قد يؤثر على أداء النظام ككل.

1.8.5. الأجهزة المتجمعة (**Clustered Systems**)

هي مجموعة من الأجهزة المتواجدة في مكان واحد والمتصلة مع بعضها البعض بشبكة محلية سريعة وغالبا ما تكون متشابهة في المكونات المادية ونظم التشغيل التي تديرها. تتشارك في التخزين (قد يكون هنالك جهاز واحد بقرص صلب والبقية بدون مثلا). تستخدم لتنفيذ البرامج الضخمة والتي تحتاج وقت كبير حيث يقسم التطبيق إلى أجزاء صغيرة تنفذ في أجزاء التجمع، الشكل (1-6), الفرق بين النظم الموزعة والحاسبات المجتمعة التباين في الأجهزة والبعد الجغرافي.



شكل رقم (1-6): تجمع أجهزة.

1.8.6. الأجهزة ذات الزمن الحقيقي (Real-time)

هي حواسيب موجودة في أجهزة تحكم مثل:

- أجهزة تجميع السيارات.
- الماكينات.
- عملية الطيران .
- إطلاق الصواريخ.
- النظم الطبية.
- الإنسان الآلي (Robotics)، كما في الشكل (1-7).

تتصف نظم التشغيل التي تدير مثل هذه الحواسيب بقيد زمني، حيث لابد من أن يتم التنفيذ في فترة زمنية محددة، لان التنفيذ مرتبط بعمل يجب أن ينجز في وقت معين وقد يتسبب في تلف ما إن نفذ في وقت متأخر أو متقدم عن الزمن المحدد له.



شكل رقم (1-7): إنسان آلي استخدم لاكتشاف كوكب المريخ.

1.8.7. الأجهزة الكفية (**hand held**)

هذه الأجهزة صغيرة غالبا ما تحمل في الجيب أو في الكف ومن هنا جاءت تسميتها بالأجهزة الكفية (hand held) أو الجيبية (pocket PC)، وتتصف بالآتي:

- حجم صغير.
- ذاكرة محدودة.
- معالج بطيء.
- شاشة صغيرة.
- بطارية محدودة التشغيل.

أمثلة لها: I-mate, Nokia E95, Toshiba e750

نظام التشغيل لهذا النوع من الأجهزة يدعم المحادثات التلفونية والتصوير الرقمي وتبادل البيانات مع الأجهزة الأخرى عبر الشبكات اللاسلكية والبلوتوث. كما يهدف نظام التشغيل هنا على المحافظة على البطارية، والتعامل مع الشاشات الصغيرة وتوفير طرق سهلة للتعامل مع الأوامر (في شكل نقرات دون الحاجة لاستخدام لوحة المفاتيح التي يكون التعامل معها صعبا لصغر حجمها).

أمثلة لنظم التشغيل التي تعمل على هذا النوع من الأنظمة:

Palm, Windows Mobile, Symbian



شكل رقم (1-8): حاسب جيبي.

1.8.8. الأنظمة المضمنة (Emedded Systems)

هي حاسبات ذات أغراض معينة، صممت لتقوم بوظيفة واحدة، وغالبا ما تكون مضمنة داخل جهاز تتحكم في عمله، مثل التلفزيون، السيارة، المايكرويف، مشغل MP3، الموجهات (routers)، إشارات المرور، مسجلات DVD، وغيرها. بعض هذه الحاسبات تحتوي على نظام تشغيل ينفذ أعمال تحكمية.

الفرق بين الحاسبات الكفية والمضمنة أن الأخيرة لا يمكن إضافة برامج جديدة إليها، فهي تاتي ببرامجها مخزنة في ذاكرة ROM من الشركة.

1.8.9. أنظمة البطاقات الذاكية (Smart card Systems)

البطاقات الذكية هي كروت بلاستيكية بحجم البطاقة الإئتمانية مزودة برقاقة الكترونية صغيرة لها القدرة على معالجة المعلومات، وهذا يعني انها تمتلك القدرة على استقبال البيانات او المدخلات ومعالجتها من خلال البرمجيات المثبتة على هذه الشريحة. تمتلك هذه البطاقة ذاكرة وطاقة تحويل إلكتروني تتخذ أساليب دفع مختلفة مثل الدفع الإلكتروني كبطاقة إئتمانية، التأمين الصحي (شكل رقم 1-9)، التعرف على الشخصية (بدل البطاقة الشخصية) وغيرها من المهام.

شريحة البطاقة الذكية تستطيع معالجة البيانات بالإضافة إلى تخزينها. نظام التشغيل المصمم لهذا النوع من الأجهزة يعتبر صغير جدا. تنتشر البطاقة الذكية في أوربا أكثر من إنتشارها في أمريكا.



شكل رقم (1-9): بطاقة ذكية تستخدم للتأمين الصحي بفرنسا

1.8.10. أنظمة المخدمات **Server Systems**

هي أجهزة سريعة وقوية تستخدم لتوفير خدمات لبقية الأجهزة في الشبكة. هنالك نظم تشغيل خاصة بهذه الاجهزة مثل Windows Advanced Server، Unbuntu Server. حيث تدعم هذه النظم أهم صفة تتصف بها المخدمات وهي إتاحة المشاركة بين المستخدمين وخدمة أكبر عدد منهم في وقت واحد. مثلا مخدم الويب (web server) يسمح لعدد كبير من المتصفحين من الاتصال وتصفح المواقع.

الأنظمة أعلاه متنوعة في الشكل والغرض وبالتالي لابد من إدارة كل نوع بطريقة مختلفة. وبالتالي يختلف الهدف من بناء نظام تشغيل باختلاف هذه الأنظمة.

1.9. ملخص

لقد بدأت الحاسبات من غير نظم تشغيل لذلك كان استخدامها محصورا على المهندسين المختصين، ثم تطورت صناعة المكونات المادية والبرامج إلى أن وصلنا لنظم تشغيل تمكن أي شخص عادي من التعامل مع الحاسب بكل سهولة ويسر.

التطور في نظم التشغيل جعل تفاصيل المكونات المادية المعقدة مخفية عن المستخدم وعن التطبيقات فيما يسمى الآلة الافتراضية، فالمستخدم يتعامل مع برمجيات تمثل نظام التشغيل وكأنها الحاسب، في بيئة سهلة الاستخدام، بينما تقوم طبقات نظام

التشغيل بالعمل المعقد والتعامل مع المكونات المادية دون أن يرى المستخدم هذه التفاصيل أو يهتم بها. التباين في أنواع الأجهزة والأغراض التي من اجلها صممت، ولدت تباين في نظم التشغيل التي صممت لها، فهنالك نظم تشغيل لكل نوع، تتناسب معه وتديره بالطريقة المثلى.

1.10. تمارين محلولة

أختار الإجابة الصحيحة (قد تكون هنالك أكثر من إجابة صحيحة)

تنقسم نواة نظام التشغيل إلى 5 أجزاء رئيسية منها:

- مدير الشاشة
- مدير الأجهزة. (1)
- مدير القرص الصلب.
- مدير القرص الضوئي

ما الذي يعتبر واجة من واجهات نظام التشغيل:

- سطح المكتب (1)
- المتصفح
- واجهة نداء النظام (1)
- لا شيء مما ذكر.

مثال لنظام تشغيل أحادي البرامج :

- DOS (1)
- ويندوز XP
- لينكس
- ماك

من نظم التشغيل التالي:

- رينكس

- زينكس
- ماك (1)
- لا شيء مما ورد

يتميز تعدد المعالجات بأنه:

- كثير التكلفة
- سريع (1)
- معقد
- إذا تعطل معالج فسيتوقف النظام

من الميزات التي لا توجد في الأنظمة الموزعة :

- التشارك في الموارد.
- زيادة سرعة التنفيذ.
- توزيع الحمل بين هذه الحاسبات
- لا شيء مما ذكر (1)

الأجهزة الكفية (hand held) تتصف بالآتي:

- حجم صغير. (1)
- ذاكرة كبيرة.
- معالج سريع.
- شاشة صغيرة. (1)

من أمثلة نظم تشغيل الحاسبات الكفية :

- بالم Palm (1)
- ينكس Unix
- سيمبيان Symbian (1)
- زينكس Zenex

الفرق بين الحاسبات الكفية والمضمنة أن الأجهزة المضمنة:

- صغيرة المعالج
- ذاكرتها محدودة
- لا يمكن إضافة برامج جديدة إليها (1)
- برامجها مخزنة في ذاكرة ROM من الشركة. (1)

أجب بنعم أو لا (مع تصحيح الإجابة الخاطئة)

لا يؤثر الاتصال بين أجزاء النظام الموزع على أداء النظام. لا ، يؤثر

تعتبر الأنظمة المجمعة (cluster) مجموعة من الأجهزة المتواجدة في أماكن مختلفة والمتصلة مع بعضها البعض بشبكة عالمية. لا ، متواجدة في مكان واحد ومرتبطة بشبكة محلية.

في تعدد المعالجة (multiprocessor) يمكن تعريف المعالجات المتماثلة بأنها مجموعة من المعالجات بحيث يكون فيها معالج رئيسي واحد يتحكم في النظام وينفذ مهمة رئيسية كبيرة، بينما بقية المعالجات تنفذ ما يأمرها به هذا المعالج الرئيسي. لا ، المعالجات غير المتماثلة.

تعمل الحاسبات المركزية بنظام التقسيم الزمني (Round Robin). نعم

الفرق بين النظم الموزعة والحاسبات المجمعة التباين في الأجهزة والبعد الجغرافي. نعم

الأجهزة التي تتكون من معالج واحد متعدد النواة (multi-core)، يمكن اعتبارها أجهزة متعددة المعالجات. نعم

تعمل نظم التشغيل القديمة مثل DOS بنظام مستخدم واحد مهام متعددة (Single user Multi-task). لا ، مستخدم واحد مهمة واحدة single user single task

تعمل ويندوز فيستا بنظام مستخدم واحد مهام متعددة (Single user Multi-task).
نعم

البطاقات الذكية هي كروت بلاستيكية بحجم البطاقة الائتمانية لا تحتوي على معالج. لا، بل تحتوي على معالج.

تعمل نظم التشغيل القديمة مثل DOS بنظام مستخدم واحد مهام متعددة (Single user Multi-task). F

تعتبر ويندوز فيستا مناسبة للمخدمات (servers). F

قديما كانت تستخدم البطاقة المثقبة لتثقيب المخرجات F

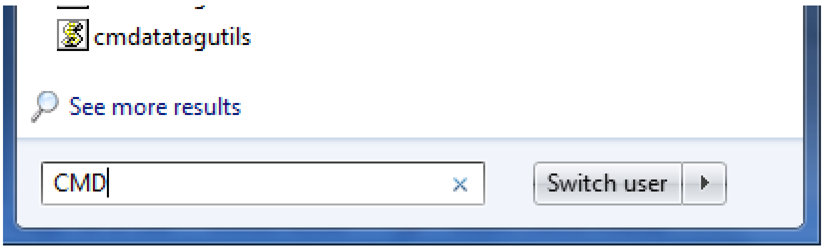
نظام التشغيل هو برنامج يدير بقية البرامج T

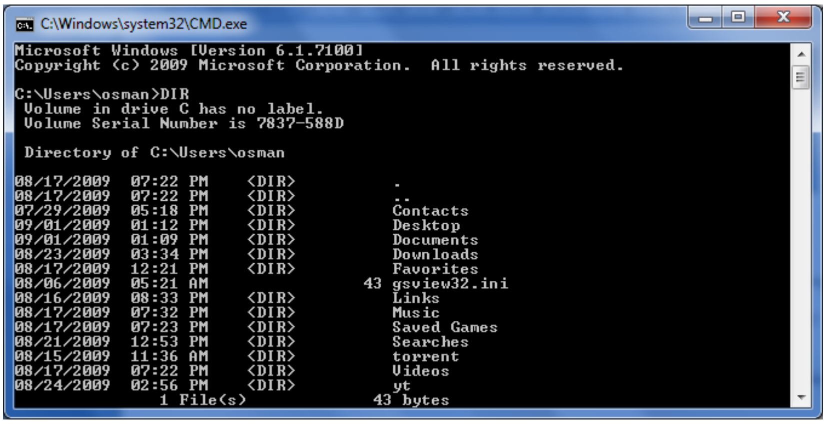
يمكن أن يتعامل المستخدم مع البرامج التطبيقية وموارد الحاسب من خلال سطح المكتب في ووينِدوز T
ليس من الضروري أن أتعامل مع نواة نظام التشغيل. T

1.11. تمارين غير محلولة

1. ما هي صفات الحاسبات القديمة ؟
2. هل يمكن استخدام الحاسب بدون نظام تشغيل (وضح) ؟
3. عرف نظام التشغيل ؟
4. كيف يدير نظام التشغيل موارد الحاسب ؟
5. ماذا نقصد بإن نظام التشغيل برنامج تحكمي ؟
6. أذكر خمسة أمثلة لنظم تشغيل ؟
7. ما هي أهداف نظام التشغيل ؟
8. أذكر ثلاث أمثلة لأنظمة حاسوب مختلفة ؟
9. ما هي مميزات الأنظمة الموزعة ؟
10. ما هو الفرق بين الأجهزة المجمعة والنظم الموزعة ؟
11. ما الفرق بين تعدد المعالجات والأنظمة الموزعة ؟
12. ما هي أهم صفة في أنظمة الزمن الحقيقي (real-time) ؟
13. أذكر ثلاث أمثلة لأجهزة تحتوي حاسوب مضمنة ؟
14. ما الفرق الرئيسي بين الأنظمة الكفية والأنظمة المضمنة ؟
15. ما هي صفات الأجهزة الكفية ؟
16. ماالفرق بين الحواسيب المتماثلة والغير متماثلة في تعدد المعالجات ؟
17. تتكون واجهات نظام التشغيل من ثلاث مكونات، ما هي؟
18. مالفرق الرئيسي بين تعدد البرامج والمشاركة الزمنية؟

19. وضح الفرق بين نظم التشغيل أحادية البرامج ونظم التشغيل متعددة البرامج مع ذكر مثال لنظام تشغيل من كل نوع؟
20. هل لكل نظم التشغيل واجهة مستخدم رسومية ؟
21. كم نداء نظام يوجد في ويندوز XP، وهل هي نفس نداءات النظام لويندوز فيستا ؟
22. أذكر أشهر نداءات النظام الموجودة في ويندوز XP ، ثم وضح كيف يمكن استخدامها داخل برامج جافا وبرامج C++ ؟
23. هل هنالك فرق بين نداءات نظام ويندوز وWindows API (وضح) ؟
24. قم بتشغيل اسطوانة أوبونتو الحية (live CD)، لتجربة نظام تشغيل أوبونتو دون تحميله على جهازك، قارن بينه وبين ويندوز XP أو بينه وبين ويندوز فيستا؟
25. أكتب الأمر cmd في قائمة ويندوز (تشغيل run)، لتنتقل إلى مترجم الأوامر (command line interpreter)، وأكتب بعض أوامر DOS مثل cls, dir, time, date, ver لتعرف كيف يمكن للمستخدم اصدار اوامر نصية إلى نظام التشغيل ؟





26. كيف اكتب أوامر نصية في توزيعة لينكس (مثل توزيعة أوبونتو) ؟ باستخدام الطرفية (terminal).

27. قم بفتح الطرفية في أوبونتو ونفذ عليها بعض الأوامر النصية مثل؟

الباب الثاني:
الحاسب وبنية نظام التشغيل

الباب الثاني

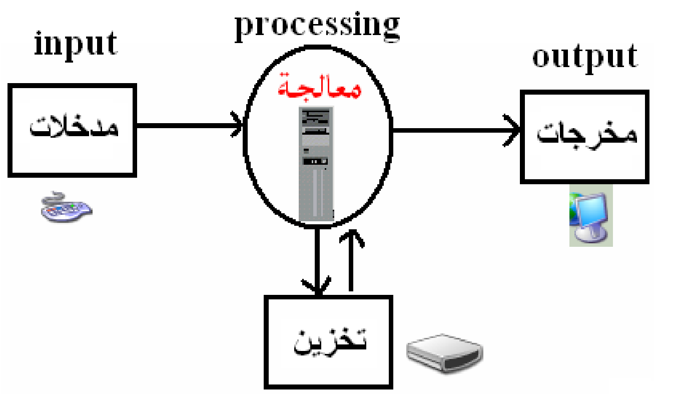
الحاسب وبنية نظام التشغيل

من المهام الرئيسية التي يقوم بها نظام التشغيل هي إدارة المكونات المادية للحاسب الآلي، لذلك لن نفهم عمل نظام التشغيل ما لم نفهم المكونات المادية التي يديرها. لهذا السبب سندرس هنا المكونات المادية وأجزاء نظام التشغيل التي تدير هذه المكونات، وكيف يوفر نظام التشغيل واجهات تمكن التطبيقات والمستخدم من التعامل مع هذه المكونات بالصورة المثلى.

2.1. عملية الحوسبة (*computing*)

تتم معظم عمليات الحوسبة في أربعة خطوات رئيسية، الشكل (2-1)، هي:

- إدخال بيانات.
- معالجة المدخلات.
- إخراج معلومات (نتيجة المعالجة).
- الاحتفاظ بالمدخلات و/أو بالنتائج (المخرجات) لاستخدامها فيما بعد (التخزين الدائم).

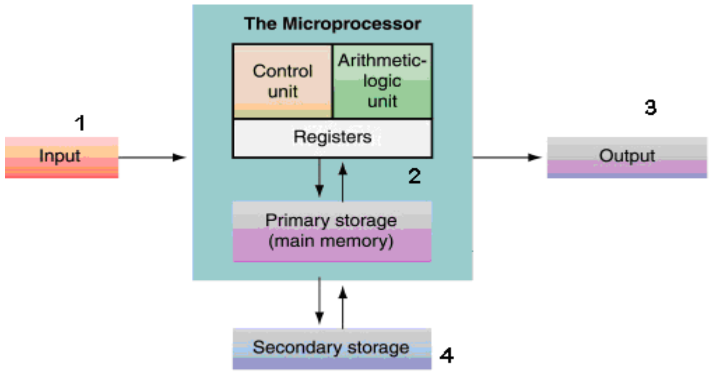


شكل رقم (2-1): عملية الحوسبة (computing).

تتكرر هذه الخطوات الأربعة باستمرار. ونجد أن المكونات المادية والبرامج تدور في محور هذه الخطوات. فلإدخال معلومة إلى الحاسب سنحتاج جهاز ونحتاج برنامج يدير هذا الجهاز. ولمعالجة المدخلات (تنفيذها) لابد من جهاز للتنفيذ وبرنامج يدير هذا الجهاز. ولإخراج النتيجة لابد من جهاز يخرج النتائج وبرنامج يقوم بإدارة هذا الجهاز. ولتخزين المدخل أو النتيجة لابد من جهاز تخزين وبرنامج يدير عمليات التخزين هذه.

إذن يمكن تقسيم المكونات المادية حسب الخطوات أعلاه إلى أربعة أجزاء رئيسية (كما مبين في الشكل (2-2))، وهي:

1. أجهزة الدخل (Input devices).
2. أجهزة المعالجة (المعالج والذاكرة (Processor & main memory)).
3. أجهزة الخرج (Output devices).
4. أجهزة التخزين الثانوية (Secondary storage).



شكل رقم (2-2): أجزاء الحاسب.

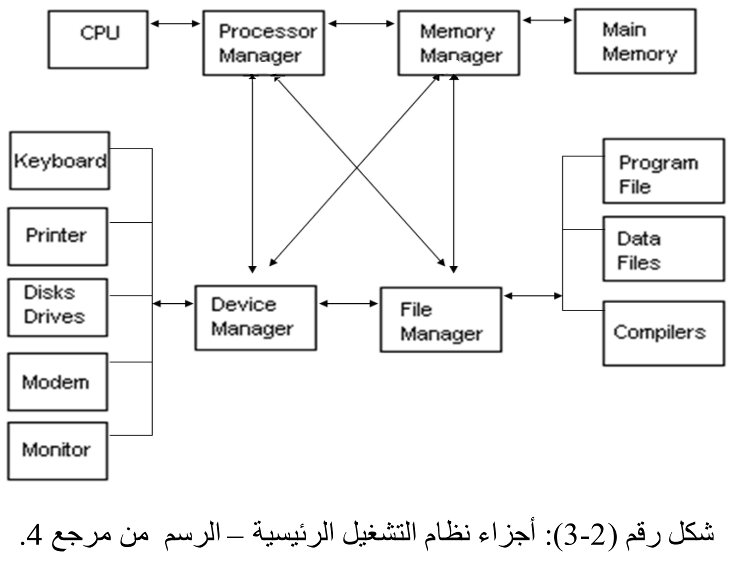
نستخدم أجهزة الدخل لإدخال المعلومات والأوامر للحاسب، فتخزن مؤقتا في الذاكرة الرئيسية ثم تتم معالجتها بواسطة المعالج، ثم تخرج النتائج بالذاكرة الرئيسية و تظهر غالبا على أجهزة الخرج، ويمكن الإحتفاظ بنسخة من المدخلات/المخرجات في القرص الصلب أو أي جهاز تخزين ثانوي (حفظ دائم).

البرامج التي تدير أجزاء الحاسب هي أجزاء نظام التشغيل وهي مقسمة كالتالي:

- مدير الأجهزة: يدير أجهزة الدخل والخرج.
- مدير العملية: يدير المعالج ويقوم بتشغيل البرامج عليه.
- مدير الذاكرة: يدير الذاكرة الرئيسية.
- مدير الملفات: يقوم بإدارة الملفات وطرق تخزينها.
- مدير الشبكة: يدير موارد الشبكات والتي تتعلق بالاتصالات الخارجية.

كل جزء من أجزاء نظام التشغيل أعلاه مكلف بإعمال على المكونات المادية التي يديرها، مثل:

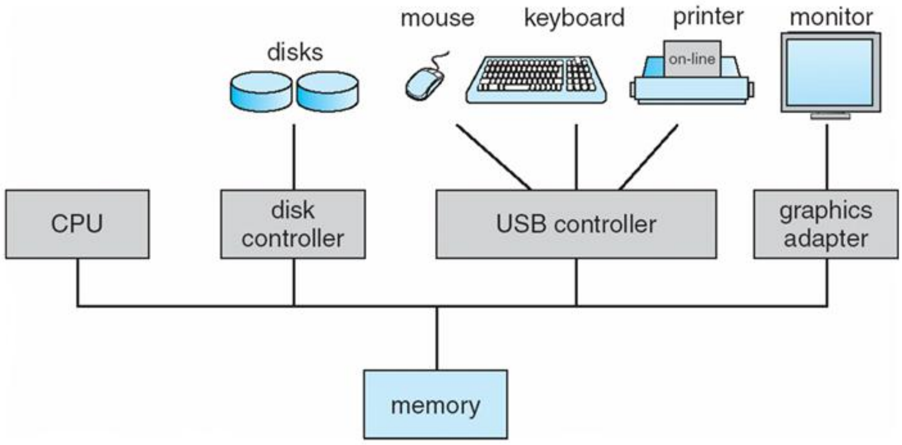
- مراقبة المكونات المادية بصورة مستمرة.
- تحديد وتنفيذ السياسات التي تحدد من يستخدم ماذا؟ متى وكيف ؟
- حجز المكونات المادية في الوقت المناسب.
- تحرير المكونات المادية في الوقت المناسب.



شكل رقم (2-3): أجزاء نظام التشغيل الرئيسية – الرسم من مرجع 4.

2.2. أجزاء الحاسب

تتكون الحاسبات الحديثة من معالج أو أكثر، متحكمات، ناقل يربط المعالج والمتحكمات بذاكرة مشتركة، الشكل (2-3).



شكل رقم (2-4): نموذج لحاسب شخصي – الرسم من مرجع 2.

2.2.1. المتحكم (controller)

كل جهاز مرتبط بمتحكم (controller)، حيث نجد أن:

- المتحكم يعمل دوما في خدمة الجهاز.
- قد يكون الجهاز معقد جدا فيوفر المتحكم واجهة بسيطة يتعامل بها نظام التشغيل مع الجهاز.
- يوجد بكل متحكم ذاكرة تسمى الخازن (buffer) تساعد في عملية نقل البيانات بين الجهاز والذاكرة الرئيسية.
- يوجد في بعض المتحكمات معالج يقوم بالتعامل مع الجهاز وتفاصيله المعقدة، مثلا يقوم معالج متحكم الشاشة بحساب القيم والنقاط التي يجب رسمها على الشاشة، بينما يرسل المعالج الرئيسي المعلومات وأين يجب أن تظهر في الشاشة.

- يمكن أن تعمل المتحكمات مع المعالج في وقت واحد بالتوازي (concurrently).
- متحكم الذاكرة الرئيسية يقوم بتنظيم وصول المتحكمات الأخرى والمعالج للذاكرة بصورة تزامنية.
- خازن المتحكم يساعد في تسريع نقل البيانات بين الذاكرة الرئيسية والجهاز. فالجهاز غالبا يرسل البيانات في شكل بايتات، تجمع في ذاكرة المتحكم حتى تمتلئ ثم ترسل دفعة واحدة إلى الذاكرة، بدلا من إرسالها بايت بعد بايت.

2.3. المقاطعات (***Interrupts***)

المعالجات الحديثة توفر طريقة تمكن المكونات المادية (أجهزة الدخل/الخرج) من إرسال إشارة للمعالج، تعتبر هذه الإشارة طلب مقاطعة للمعالج. هذه المقاطعة توقف عمل المعالج مؤقتا، لينفذ خدمة مطلوبة (interrupt service routine). هنالك مجموعة من الخدمات تختلف بإختلاف نظام التشغيل، وهي عبارة عن دوال خاصة بالمقاطعات، فكل مقاطعة تقابلها مجموعة من الدوال، عندما يتم إرسال المقاطعة إلى المعالج يوقف ما كان يعمل فيه، ثم ينفذ دالة معينة من دوال المقاطعة (تحدد الدالة برقم في مسجل معين)، ثم يرجع المعالج مرة أخرى ليواصل ما أوقفه قبل إستلامه للمقاطعة.

قد تصدر مقاطعات من أكثر من جهاز، وعلى المعالج أن يرد عليها جميعها بالطريقة المناسبة وفي وقت قصير. هنالك أنواع مختلفة من المقاطعات :

- خارجية وتصدر من أجهزة الدخل/الخرج.
- داخلية قد تصدر من المؤقت (timer)، أو نتيجة عطل في مكون مادي (hardware failure)، أو من برنامج.

مثلا إذا كان هنالك برنامج تحت التنفيذ (P1 مثلا) إحتاج لمعلومة من جهاز دخل، فسيقوم المعالج بإرسال طلب المعلومة إلى جهاز الدخل المعني، ثم يتحول لتنفيذ برنامج آخر (P2 مثلا). عندما يصبح جهاز الدخل مستعد لنقل البيانات، سيرسل إشارة مقاطعة إلى المعالج يخبره فيها بأن البيانات جاهزة. سيرد المعالج على المقاطعة بتوقيف البرنامج الثاني (P2)، وينتقل لتشغيل دالة المقاطعة (interrupt handler) التي تقوم

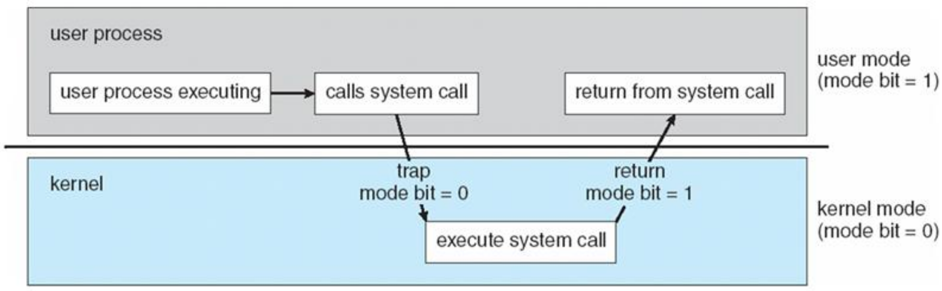
بإكمال نقل البيانات. في هذا اللحظة يمكن للمعالج أن يواصل تشغيل البرنامج الأول (P1).

2.4. الوضع الثنائي ***dual mode***

للتنفيذ الصحيح لنظام التشغيل وفصله عن برامج المستخدم لابد من طريقة للتمييز بين البرامج التابعة له وبرامج المستخدم. ذلك لأن برامج نظام التشغيل لها صلاحيات أعلى من التي لبرامج المستخدم.

هنالك خانة (بت bit) تضاف للمكونات المادية، إذا كانت قيمة هذه البت صفر فهذا يعني أن البرنامج في وضع النواة (kernel mode)، أما إذا كانت البت تحتوي على واحد فهذا يعني أن البرنامج في وضع المستخدم (user mode). بهذه الطريقة نستطيع معرفة في أي وضع يتم تنفيذ البرامج.

أحيانا قد يحتاج برنامج ما، إلى استدعاء خدمة من نظام التشغيل (نداء نظام system call)، في هذه الحالة لابد لهذا البرنامج من أن يتغير وضعه من وضع المستخدم إلى وضع النواة، ثم بعد إكتمال تنفيذ نداء النظام سيرجع البرنامج إلى وضع المستخدم مرة أخرى، الشكل رقم (2-).



شكل رقم (2-): التحول بين وضع المستخدم ووضع النواة (الرسم من مرجع 2)

2.5. المؤقت ***timer***

من مهام نظام التشغيل التحكم في المعالج ومنع برامج المستخدمين من الاستئثار بهذا المورد الهام والعمل لمدة طويلة داخل المعالج. مثلا إذا كان هنالك برنامج ينفذ في تكرار غير منتهي (infinite loop) أو استدعى دالة خدمة ولم يُعيد

السيطرة لنظام التشغيل، فهذا يسبب إهدار لزمن المعالج. هنا لابد لنظام التشغيل من آلية تمكنه من توقيف مثل هذه البرامج، وكان المؤقت (timer) هو الحل.

2.5.1. كيف يعمل المؤقت:

- قبل تشغيل برنامج المستخدم يتأكد نظام التشغيل من أن المؤقت مرتبط مع مقاطعة.
- يتم إعداد المؤقت ليصدر المقاطعة بعد فترة محددة (لا تزيد عن ثانية).
- يكون هنالك عداد وساعة لحساب هذه الفترة.
- يقوم نظام التشغيل بتجهيز العداد.
- كل دقة ساعة تنقص العداد.
- عندما يصل العداد صفر تصدر المقاطعة وينتقل التحكم تلقائيا إلى نظام التشغيل.
- لنظام التشغيل حق التصرف في إعتبار أن هذه المقاطعة خطأ جسيم (fatal error) أو قد يعطي البرنامج مهلة أكثر (زيادة زمن).

2.5.2. مثال

إذا كان لدينا برنامج يحتاج 7 دقائق من التنفيذ، فسيتم وضع 420 (7*60) في العداد، وكلما تمر ثانية سيرسل المؤقت مقاطعة وينقص العداد بواحد، ويظل التحكم لدى البرنامج (ما دام محتوى العداد موجب)، إذا وصل العداد إلى صفر سيقوم نظام التشغيل بإنهاء البرنامج.

2.6. هرمية الذاكرة

تتكون هرمية الذاكرة من التالي:

- المسجلات registers.
- الذاكرة المخبأة (الكاش) cache.

- الذاكرة الرئيسية (الرام) main memory.
- الأقراص الممغنطة magnetic disk.
- الأقراص الضوئية optical disks.
- الأشرطة الممغنطة magnetic tape.

هنالك عوامل تؤثر في هرمية الذاكرة هي :

1. التطاير.
2. السرعة.
3. السعة.
4. السعر.

2.6.1. التطاير (volatility)

تنقسم هرمية الذاكرة إلى قسمين رئيسيين هما:

- أعلى الهرم: يخزن البيانات تخزيناً مؤقتاً يزول عند إغلاق الحاسب (متطايرة volatile)
- أسفل الهرم: يخزن البيانات تخزيناً دائماً لا يزول (غير متطايرة non-volatile).

2.6.2. السرعة

النوع الأول يخزن البيانات في شكل نبضات إلكترونيه (مثل الرام)، بينما النوع الثاني يخزن البيانات مغنطيسياً أو ضوئياً ويعمل بطريقة ميكانيكية (مثل القرص الصلب والأقراص الضوئية والشرائط الممغنطة). لذلك نجد أن سرعة الوصول للبيانات في الأول عالية جداً مقارنة مع سرعتها في النوع الثاني.

مثلا الزمن المستغرق لقراءة بايت:

- من الذاكرة المخبأة (الكاش) هو 10 نانو ثانية (ns).
- من الذاكرة الرئيسية (الرام) هو 100 نانو ثانية (ns).
- من القرص الصلب 15 ملي ثانية (ms).

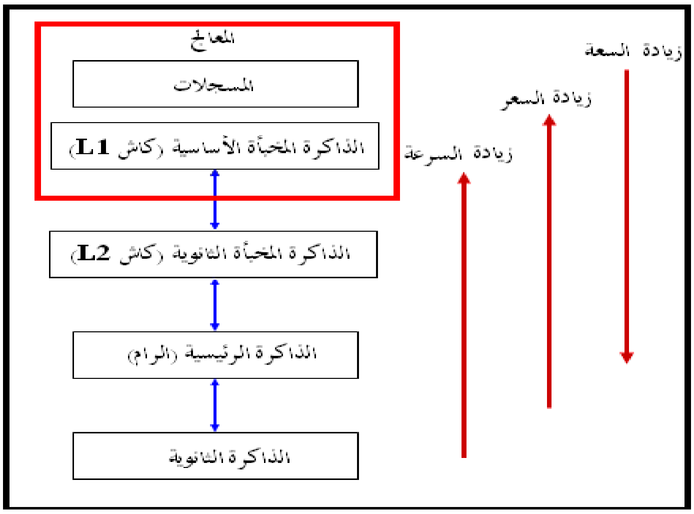
2.6.3. السعة والسعر

النوع الأول صغير الحجم ذلك لأن صناعته مكلفة إذا زدنا حجمه يصبح سعره باهظاً وبالتالي يؤثر في سعر الحاسب ككل. أما النوع الثاني كبير الحجم لان صناعته غير مكلفة، فهو رخيص جداً مقارنة بالأول.

مقارنة سعر الميغابايت (MB):

- للذاكرة المخبأة (الكاش) يكلف حوالي 200 دولار أمريكي.
- للذاكرة الرئيسية يعادل حوالي 10 دولارات أمريكية.
- للقرص الصلب يساوي حوالي 0.001 دولار أمريكي.

إذن هنالك أربع عوامل تميز بين أنواع الهرمية هي التطاير، السعة، السعر، والسرعة كما مبين في الشكل (2-10).



شكل رقم (2-10): التدرج الهرمي للذاكرة.

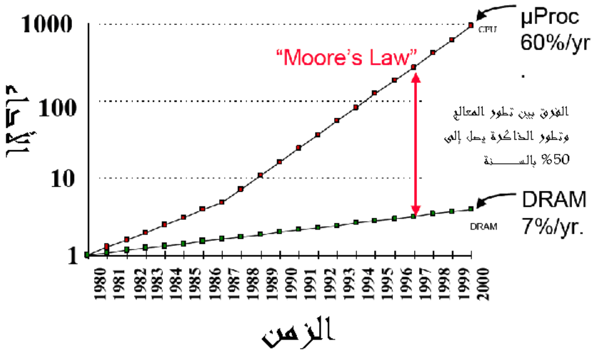
2.6.4. الذاكرة المخبأة (الكاش cache)

الغرض من الذاكرة المخبأة هو تمكين المعالج من الوصول لبيانات الذاكرة الرئيسية بشكل سريع جدا.، لذلك فالذاكرة المخبأة تؤثرا تأثيرا كبيرا على أداء المعالج. بعض المخابئ تعمل بسرعة المعالج وتسمى L1 بينما المخابئ الأخرى تعمل بنصف السرعة أو اقل وتسمى L2. المخبأ الصغير الذي يعمل بسرعة كاملة قد تكون أكثر نفعا من مخبأ كبير يعمل بنصف سرعة المعالج.

2.6.4.1. كيف تزيد الذاكرة المخبأة سرعة المعالج؟

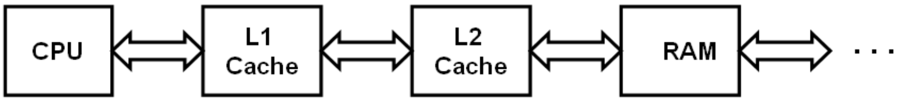
إن سرعة الوصول للبيانات في القرص الصلب (الذاكرة الثانوية) أبطأ ملايين المرات من سرعة الوصول للبيانات في الرام، لذلك تعتبر الذاكرة الرئيسية (الرام) متطلباً أساسياً لتنفيذ البرامج، فلا يمكن تنفيذ برنامج ما لم يحمل بها.

في بدايات تصنيع الحاسب كانت سرعة الذاكرة الرئيسية يساوي سرعة المعالج ولكن بمرور الزمن تطورت صناعة المعالجات وصارت سرعتها عالية جدا مقارنة مع التطور في صناعة الذاكرة الرئيسية، فلم تزيد سرعة الذاكرة بنفس نسبة زيادة سرعة المعالج مما ولد فرق كبير بين السرعتين، الشكل(2-12). وبما أن سرعة الحاسب ككل مرتبطة بسرعة أبطأ جهاز به. سنجد أنه لابد من معالجة هذا الفرق بين السرعتين بتحسين أداء الذاكرة لنستفيد من سرعة المعالج. فالمعالج سيبطئ عمله ليعمل بسرعة الرام وبالتالي لن نستفيد من سرعة المعالج ولن يظهر هذا في أداء الحاسب ككل.



شكل رقم (2-12): تزيد سرعة المعالج بنسبة 60% بينما تزيد الرام بنسبة 7% فقط سنويا.

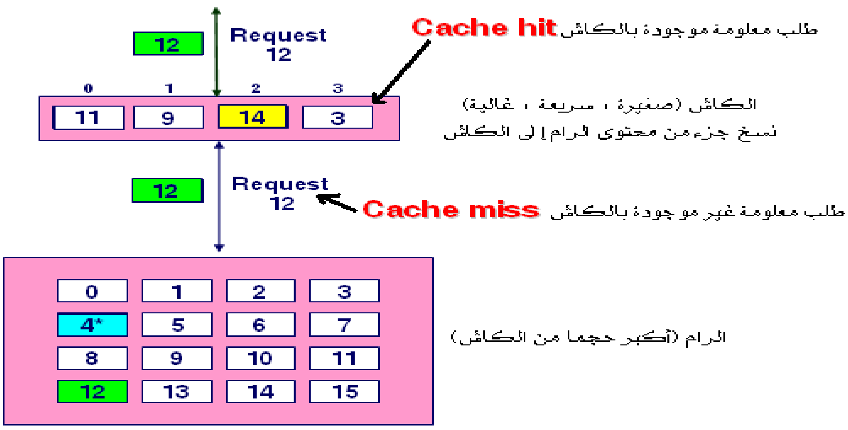
هنا جاءت الذاكرة المخبأة (الكاش) بين المعالج والرام لتحل هذه المشكلة، حيث تعتبر الكاش ذاكرة سريعة تعمل بسرعة المعالج وبالتالي لن يتأثر أداء المعالج حين يتعامل مع الكاش، شكل (2-13).



شكل رقم (2-13): الكاش كوسيط بين الرام (RAM) والمعالج (CPU).

طريقة تعامل المعالج مع الكاش تتم كالتالي:

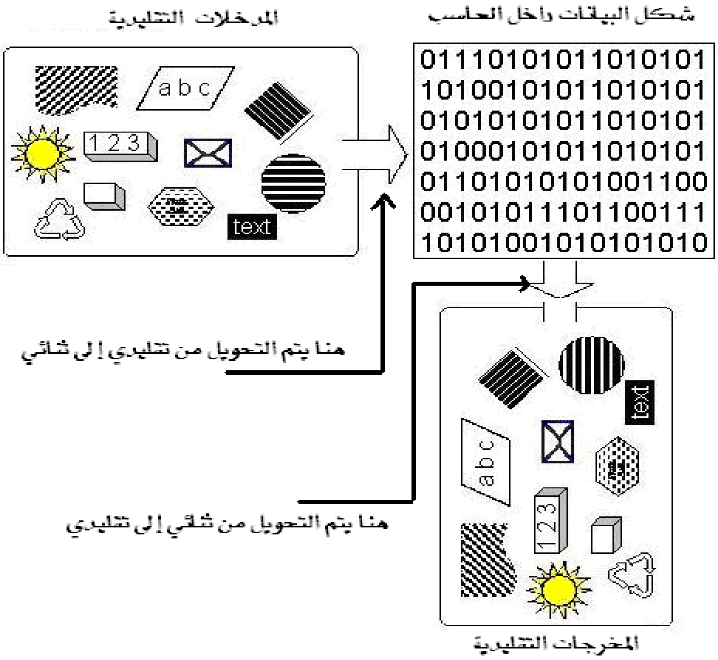
- يتم تحميل البرنامج من القرص الصلب إلى الرام.
- يتم تحميل جزء من البرنامج الموجود في الرام إلى الكاش.
- يطلب المعالج ما يريد من الكاش(يحضر بياناته من الكاش إلى المسجلات) للعمل عليها.
- ستبحث الكاش عن البيانات التي يريدها المعالج بداخلها وإذا لم تجدها ستقوم بإحضارها من الرام، الشكل (2-14).



شكل رقم (2-14): تعامل الكاش مع الرام.

2.7. التخزين الرقمي للبيانات

عند صنع أول حاسب كانت المدخلات والمخرجات هي أرقاما ثنائية لأن الحاسب لا يتعامل إلا مع الأرقام الثنائية، لذلك كان من العسير التعامل مع الحاسب. ثم مع التطور وصلنا لطرق تمكننا من التعامل مع الحاسب بطريقتنا التقليدية البسيطة (حروف وصور وأصوات وغيرها)، لكن الحاسب ما زال يتعامل مع كل شيء ثنائيا (صفر أو واحد).



شكل رقم (2-5): تحويل البيانات من الشكل التقليدي إلى الشكل الرقمي والعكس.

يتم ذلك عن طريق أجهزة وبرامج تقوم بتحويل البيانات من الشكل التقليدي إلى الأرقام الثنائية عند الإدخال، فنقوم نحن بالإدخال بالصورة التقليدية وتقوم الأجهزة والبرامج الوسيطة بتحويل البيانات التقليدية إلى أصفار و وحائد قبل تخزينها في الحاسب وبهذا تتحول البيانات إلى الشكل الذي يفهمه الحاسب دون تدخل منا. كذلك عند ما تظهر المخرجات بالشكل التقليدي فهذا لأن هنالك أجهزة وبرامج وسيطة تحول المخرجات من أصفار و وحائد (الشكل المخزن بالحاسب) إلى الشكل التقليدي قبل

إظهارها في الشاشة أو سماعها، أنظر الشكل (2-5). أي أن كل معلوماتنا عندما تخزن في ذاكرة الحاسب تكون خليط من الوحائد و الأصفار.

يظهر هنا سؤال هام هو، كيف يميز الحاسب بين المعلومات والرموز والأشكال إذا كانت كلها خليط من الوحائد و الأصفار ؟ الإجابة باستخدام التشفير كما سنوضح في الفقرة (2.2.1).

لا تعتبر أجهزة الدخل والخرج جزء من الحاسب وإنما هي أدوات لإدخال البيانات وإظهار النتائج، فهي السبيل الذي نتعامل به مع الحاسب. هنالك بعض الحاسبات لا تحتاج هذه الأجهزة.

2.7.1. تمثيل البيانات داخل الحاسب

الرموز والأرقام والحروف التي نستخدمها في لوحة المفاتيح لها مقابل من الأرقام الثنائية (خليط من الأصفار و الوحائد)، فكل رقم وكل رمز وكل حرف في لوحة المفاتيح له قيمة ثنائية تمثله داخل الحاسب، مثلا الجدول (2-1) يوضح قيم الرموز والحروف للحاسبات التي تستخدم شفرة آسكي (ASCII)، عندما نضغط على الرمز أو الحرف في لوحة المفاتيح تخزن القيمة المقابلة له (كما في الجدول) بذاكرة الحاسب.

هنالك العديد من الشفرات المستخدمة لتمثيل البيانات في الحاسب مثل:

- شفرة EBCDIC: الشفرة الثنائية الموسعة للتبادل العشري
- شفرة آسكي (ASCII) : الشفرة الأمريكية النمطية لتبادل المعلومات.
- الشفرة الموحدة Unicode.

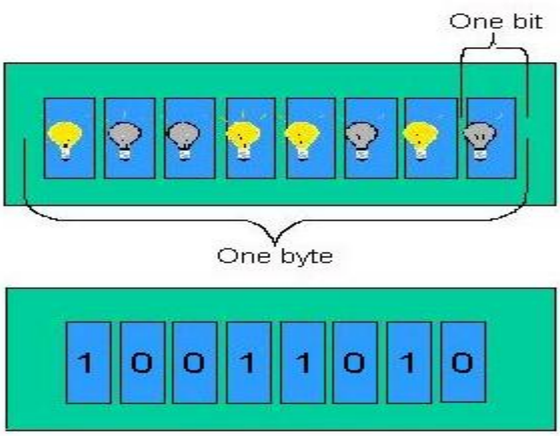
الآن معظم نظم التشغيل تستخدم الشفرة الموحدة unicode، وهي تدعم اللغة العربية، أي ان هنالك قيمة ثنائية لكل حرف من الحروف العربية.

Character	Bit pattern	Byte number		Character	Bit pattern	Byte number
A	01000001	65		¼	10111100	188
B	01000010	66		.	00101110	46
C	01000011	67		:	00111010	58
a	01100001	97		$	00100100	36
b	01100010	98		\	01011100	92
o	01101111	111		~	01111110	126
p	01110000	112		1	00110001	49
q	01110001	113		2	00110010	50
r	01110010	114		9	00111001	57
x	01111000	120		©	10101001	169
y	01111001	121		>	00111110	62
z	01111010	122		‰	10001001	137

جدول (2-1): شفرة آسكي لتمثيل الرموز.

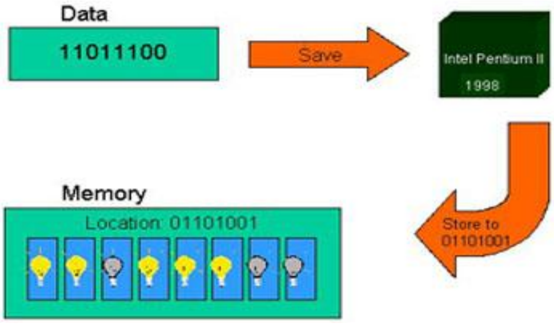
2.7.2. البت والبايت

لماذا لا يخزن الحاسب سوى الأرقام الثنائية (الصفر والواحد) ؟ لأن أجزاء الحاسب الداخلية هي عبارة عن مفاتيح تكون على واحد من حالتين، إما ON أو OFF، حيث تمثل حالة ON القيمة 1، بينما تمثل حالة OFF القيمة صفر، وكل مفتاح يمثل بت أي رقم ثنائي واحد، شكل رقم (2-6)، كل موقع تخزين في الذاكرة الرئيسية يتكون من ثمانية مفاتيح ويسمى بايت (8 بتات).، وكل بايت يستطيع تخزين رمزاَ أو حرفا واحداَ، الشكل (2-6).



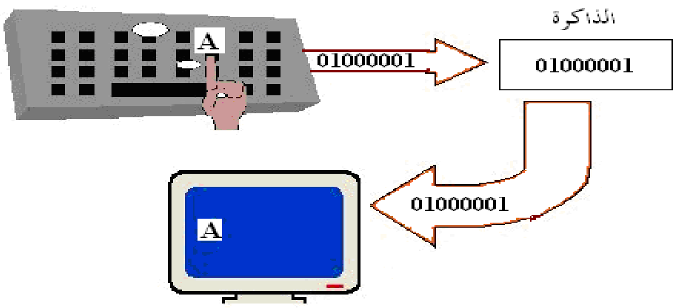
شكل رقم (2-6): كل مفتاح يمثل بت وكل 8 بتات تمثل بايت.

مثلا إذا ضغطت على زر في لوحة المفاتيح (الحرف A)، فإن القيمة الثنائية 01000001 (من الجدول (2-1)) سيتم إدخالها في ذاكرة الحاسب، كما في الشكل (2-8).



شكل رقم (2-7): كل موقع بالذاكرة الرئيسة يتألف من ثمانية مفاتيح.

إذن كل شيء داخل الحاسب هو خليط من ON و OFF أو خليط من 0 و 1.
هذا الخليط قد يكون ملف نصي مدخل عبر لوحة مفاتيح أو سورة قرآنية مسجلة بالمايكروفون، أو صورة مدخلة بواسطة الماسحة أو برنامج جافا كتبه مبرمج، أو حتى نظام التشغيل. فالكل داخل الحاسب أرقاماً ثنائية.



شكل رقم (2-8): تمثيل الحرف A داخل ذاكرة الحاسب.

هنا يبرز سؤال جديد هو كيف يميز الحاسب هذا الكم الهائل من الأصفار والوحائد، وكيف يميز بين الملفات وأنواعها ؟

الإجابة هي: جزء من نظام التشغيل يسمى مدير الملفات هو الذي يميز بين الملفات وأنواعها وأماكن تخزينها بأجهزة التخزين، وبين البرامج وأنواعها وأماكن تواجدها بالذاكرة الرئيسية.

2.8. كيف يعمل الحاسب

الآن وبعد أن عرفنا كيف تخزن البيانات في الحاسب نريد أن نعرف كيف يعمل الحاسب منذ فتحه وحتى إغلاقه.

2.8.1. الإقلاع (**Booting**)

عندما نفتح الحاسب تتم الخطوات التالية:

- أول برنامج يشتغل هو برنامج مخزن بذاكرة القراءة فقط (ROM). يقوم هذا البرنامج باختبار المكونات المادية والتأكد من أن كل أجزاء الحاسب سليمة وموصلة بطريقة صحيحة. يسمى هذا البرنامج برنامج الاختبار الذاتي (Power On Self Test (POST)).
- بعد نجاح عملية الاختبار يشتغل برنامج آخر يسمى bootstrap loader ، وهو برنامج صغير جداً وموجود بذاكرة القراءة فقط (ROM) أو أي ذاكرة غير متطايرة. مهمة هذا البرنامج هي البحث عن نظام التشغيل وتحميله بالذاكرة الرئيسية وتسليمه التحكم بالحاسب.
- هنا تنتهي مهمة هذا برنامج bootstrap loader ويستلم نظام التشغيل التحكم بالحاسب ليدير المكونات المادية ويوفر واجهة للمستخدم والتطبيقات.

2.8.2. التعامل مع نظام التشغيل

بعد تحميل نظام التشغيل وتسليمه قيادة الحاسب، سيكون هو الواجهة التي يتعامل معها المستخدم، فهو الذي يستجيب لطلباتهم ويشغل برامجهم ويقوم بكل ما يريد المستخدم. فهو مدير الحاسب وهو الذي يتعامل مباشرة مع المكونات المادية بينما نتعامل نحن وبرامجنا معه ونطلب منه ما نريد، ولديه مطلق الحرية في تنفيذ طلباتنا أو رفضها حسب إستراتيجيته وأهمية طلباتنا، أنظر الشكل (1-3).

مثلا عندما يظهر سطح المكتب في ويندوز فهذا يعني أن نظام التشغيل جاهزا لتلقي أوامر المستخدمين التي قد تكون:

- استخدام مباشرة للواجهة (سطح المكتب): مثل نسخ ملف، فتح ملفات، تخزين ملف...الخ.
- تشغيل برنامج: هنا يقوم المستخدم بإرسال طلب لنظام التشغيل بأنه يريد تشغيل برنامج معين وذلك بالنقر المزدوج أيقونة البرنامج فيقوم نظام التشغيل بالبحث عن البرنامج في مكان تخزينه الدائم وتحميله إلى الذاكرة الرئيسية وتسليمه القياد فترى برنامجك يعمل أمامك.

لابد من تحميل البرنامج للذاكرة الرئيسية قبل تنفيذه: المعلوم أن معظم برامجنا بما فيها نظام التشغيل مخزنة بالقرص الصلب، ولكن المعالج لا ينفذ أي برنامج ما لم يحمل إلى الذاكرة الرئيسية، لذلك نجد أن عملية تنفيذ البرامج تبدأ بتحميل البرنامج إلى الذاكرة الرئيسية وهنا نقول أن البرنامج جاهز للتنفيذ (ready)، بالتالي كلما كان البرنامج كبيرا كلما استغرق تحميله وقتاً أطول.

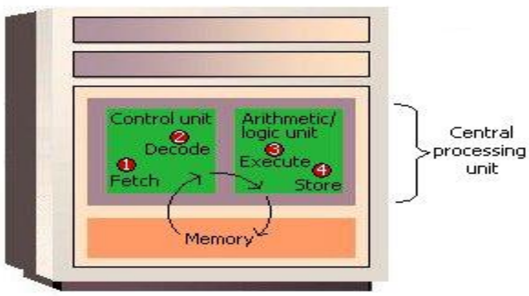
2.8.3. تنفيذ البرامج

من المعلوم أن البرنامج هو سلسلة من الأوامر (التعليمات) التي تنفذ متسلسلة (أمر تلو الآخر)، ويقوم المعالج بتنفيذ كل أمر في خمس مراحل هي:

- إحضار الأمر (fetch): هنا نحضر الأمر من الذاكرة الرئيسية ونخزنه في مسجل (داخل المعالج).
- فهم وتفسير الأمر (decode): هنا يقوم المعالج بالعمل على الأمر لفهمه ومعرفة ما إذا كان يحتاج بيانات إضافية من الذاكرة ام لا؟ مثلا إذا كان الأمر (أجمع) فهذا يعني أننا نحتاج إحضار الرقمين المراد جمعهم من الذاكرة الرئيسية.
- إحضار البيانات الإضافية (fetch operands): هنا نحضر البيانات التي يحتاجها الأمر من الذاكرة الرئيسية (إذا كان الأمر يحتاج بيانات إضافية حسب الخطوة 2 أعلاه) ونضعها في مسجلات داخل المعالج.

- التنفيذ (execute): ينفذ الأمر بواسطة وحدة الحساب والمنطق التي تعتبر جزءاً من المعالج.
- التخزين: هنا تخزن نتائج في مسجلات داخل المعالج.

يكرر المعالج الخطوات أعلاه على كل أمر من أوامر البرنامج حتى ينتهي التنفيذ، شكل (2-11).



شكل رقم (2-11): دورة تنفيذ الأوامر.

2.9. ملخص

تحدثنا في هذا الباب عن العلاقة بين عملية الحوسبة والمكونات المادية وأجزاء نظام التشغيل. ثم بينا كيف يتم تمثيل البيانات داخل الحاسب في شكل أرقام ثنائية وكيف يخفي نظام التشغيل هذه التفاصيل والتعقيدات عن المستخدم موفرا بيئة سهلة وملائمة للمستخدم.

2.10. تمارين محلولة

1. لماذا يعتبر الشريط الممغنط أبطأ أنواع الذاكرة الثانوية ؟ لأن الوصول إليه يكون تتابعي (sequential access) بينما التعامل مع القرص الصلب مثلا يكون عشوائيا (random access).

2. ما اسم البرنامج الذي يبحث عن نظام التشغيل وينفذه ؟ وأين يوجد ؟ Bootstrap loader، يوجد هذا البرنامج بذاكرة القراءة فقط (ROM).

3. ما اسم البرنامج الذي يختبر اجزاء الحاسب ويتأكد منها ؟ وأين يوجد ؟ برنامج الاختبار الذاتي (POST) ويوجد بذاكرة القراءة فقط (ROM).

4. يتكون المعالج من ثلاث أجزاء، ما هي هذه الأجزاء ؟ وحدة التحكم (CU)، وحدة الحساب والمنطق (ALU)، المسجلات (registers).

5. سرعة الحاسب ككل مرتبطة بسرعة أبطأ جهاز به (وضح بمثال) ؟ مثلا قد يدخل المستخدم حرف في كل ثانية عبر لوحة المفاتيح، بينما يحتاج المعالج 2 ميكروثانية لنقل حرف إلى الذاكرة. الثانية الواحدة تحتوي على 1000 ميكروثانية. هذا يعني أن هنالك 998 ميكروثانية تهدر من زمن المعالج في كل 1000 ميكروثانية.

2.11. تمارين غير محلولة

1. يقوم المعالج بتنفيذ كل أمر في خمس مراحل، ما هي ؟
2. ما هي العوامل التي تؤثر على هرمية الذاكرة (أربعة عوامل).
3. ما الفرق بين الكاش L1 ، الكاش L2، وأيهما أسرع ولماذا ؟
4. ما معنى cache miss ؟ و ما معنى cache hit ؟
5. ما هي مهام المتحكم (controller) ؟
6. ما الفرق بين وضع المستخدم (user mode) ووضع النواة (kernel mode)؟
7. أذكر عناصر هرمية الذاكرة مرتبة من الأبطأ إلى الأسرع ؟
8. ما هي العوامل التي تؤثر في ترتيب هرمية الذاكرة (4 عوامل) ؟
9. هنالك العديد من الشفرات المستخدمة لتمثيل البيانات في الحاسب، أذكر ثلاث منها ؟
10. أذكر باختصار خطوات إقلاع الحاسب ؟
11. تتم معظم عمليات الحوسبة في أربعة خطوات رئيسية، أذكر هذه الخطوات مع تحديد المكونات المادية التي تحتاجها كل خطوة ؟
12. لماذا لا يخزن الحاسب سوى الأرقام الثنائية (الصفر والواحد) ؟
13. أذكر أجزاء نظام التشغيل الخمسة والتي تدير موارد الحاسب ؟
14. كل جزء من أجزاء نظام التشغيل مسئول عن أعمال تجاه الموارد التي يدير، ما هي هذه الأعمال ؟
15. لماذا لم تكن هنالك ذاكرة مخبأة (كاش) بدايات الثمانينات (1980)، وما هو سبب ظهور الكاش ؟

الباب الثالث:

العمليات

الباب الثالث

العمليات (Porcesses)

إدارة العمليات (***process management***) هو جزء هام من نظام التشغيل ويعتني بكل ما يتعلق العمليات من:

- مفهوم العمليات (***porcesses***).
- الخيوط أو العمليات الخفيفة (***Threads***).
- جدولة المعالج (***CPU scheduling***).
- تزامن العمليات (***Process synchronization***).
- الإختناق بين العمليات (***Deadlocks***).

سنتحدث في هذا الباب عن العمليات، بينما سنتحدث في الباب الذي يليه (الرابع) عن جدولة المعالج ، إما الباب الخامس فسيكون مخصص للخيوط، والباب السادس سيكون عن تزامن العمليات، أما الباب السابع فقد خصصناه للإختناق.

3.1. مقدمة

قديما كانت نظم التشغيل تسمح فقط بتشغيل برنامج واحد في اللحظة الواحدة، هذا البرنامج يتحكم ويستأثر بكل موارد الحاسب من معالج وذاكرة وأجهزة دخل وخرج وملفات،... وغيرها. الآن أصبحت التشغيل الحديثة تسمح لأكثرمن برامج بأن يعمل في وقت واحد متشاركة في الموارد مما أسهم بشكل فعال في تحسين آداء الحاسب وزيادة إنتاجيته.

أيضا الإدارة الجيدة لموارد الحاسب من قبل نظام التشغيل تؤثر تأثيرا مباشرا على الآداء. أهم موارد في الحاسب هو المعالج الذي يقوم تنفيذ برامجنا.

البرامج قد تكون نظام التشغيل، المترجمات، الأوفيس، الألعاب، وبرامج المستخدم الأخرى. يتحول البرنامج إلى عملية عند ما نقوم بتشغيله.

3.2. مفهوم العملية (*Process Concept*)

البرنامج يكون في شكل ملف عندما يكون مخزن بالقرص الصلب (أو أي وسيط تخزين ثانوي) وعندما ننقر عليه نقرا مزدوجا فإننا نطلب من نظام التشغيل تنفيذه، فيقوم نظام التشغيل بتحميله من القرص الصلب (أو وسيط التخزين الموجود به مثل الفلاش أو الأسطوانة) إلى الذاكرة الرئيسية (الرام) ليبدأ التنفيذ، هنا يتغير اسم البرنامج من ملف إلى عملية.

> العملية هي برنامج شغال (تحت التنفيذ)، أحيانا نطلق عليها عمل (job) أو مهمة (task).

البرنامج هو سلسلة من الأوامر تعطى للحاسب للقيام بعمل ما. ينفذ البرنامج داخل المعالج تسلسليا، أمر تلو الآخر.

خطوات تنفيذ البرنامج:

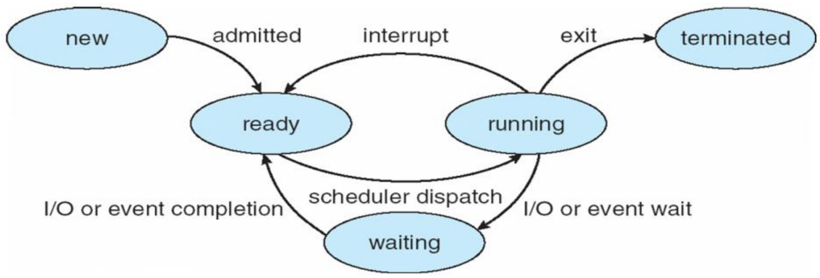
1. تحميل البرنامج في الذاكرة الرئيسية.
2. يتم وضع عنوان بداية البرنامج (عنوان أول أمر بالبرنامج) في مسجل داخل المعالج يسمى عداد البرنامج ((program counter (PC)).
3. يقوم المعالج بإحضار أول أمر بالبرنامج، هذا الأمر نعرف مكانه من خلال عداد البرامج، يحضر الأمر من الذاكرة الرئيسية ويتم تخزينه في مسجل داخل المعالج.
4. زيادة عداد البرامج ليشير إلى الأمر يليه.
5. فهم وتنفيذ الأمر الذي أحضرناه.

6. إحضار الأمر يليه (يشير له عداد البرامج)
7. زيادة عداد الأوامر ليشير إلى الأمر الذي يليه.
8. فهم وتنفيذ الامر الذي بالمعالج.
9. إنتقل إلى الخطوة 6، وهكذا نكرر هذه الخطوات إلى أن ينتهي تنفيذ البرنامج.

3.3. حالات العملية (*process states*)

تحميل البرامج في الذاكرة يجعل هذه البرامج جاهزة للتنفيذ (ready)، عند بداية تنفيذ البرنامج داخل المعالج يصبح شغال (running)، قد يستمر المعالج في تنفيذ البرنامج حتى يكتمل، وقد يوقف المعالج البرنامج الشغال (مؤقتا) لسبب ما، فيصبح البرنامج في هذه الحالة محجوز (blocked)، وقد يشتغل برنامج آخر أكثر أهمية (مثلا). إذن تحميل البرنامج بالذاكرة يسمى عملية، هذه العملية (أو البرنامج) يتغير وضعها من حال إلى حال، كما موضح أدناه:

- جديد (new): العملية تم إنشاءها وجاهزة للتحميل.
- حالة الجاهزية (ready state) : العملية تم تحميلها في الذاكرة وأصبحت جاهزة للتنفيذ.
- حالة التنفيذ (running state): العملية بدأت التنفيذ داخل المعالج (يتابع مسجل عداد البرامج تسلسل تنفيذ أوامر العملية).
- حالة الحجز أو الانتظار (blocked state or waiting): عندما يوقف المعالج عملية، تصبح هذه العملية محجوزة. يتم توقيف العملية لأسباب عدة مثل الحاجة لتشغيل عملية أخرى أكثر أهمية، أو أن العملية تنتظر حدث (event) معين لم يتم بعد، أو أن الزمن الذي خصص للعملية قد اكتمل (المشاركة الزمنية).
- الإنتهاء (terminated): هنا تكون العملية قد انتهى عملها، فتقوم بإخلاء طرفها (تحرير الموارد التي كانت تستخدمها، وإخلاء الذاكرة التي كانت تحتجزها) قبل الخروج.



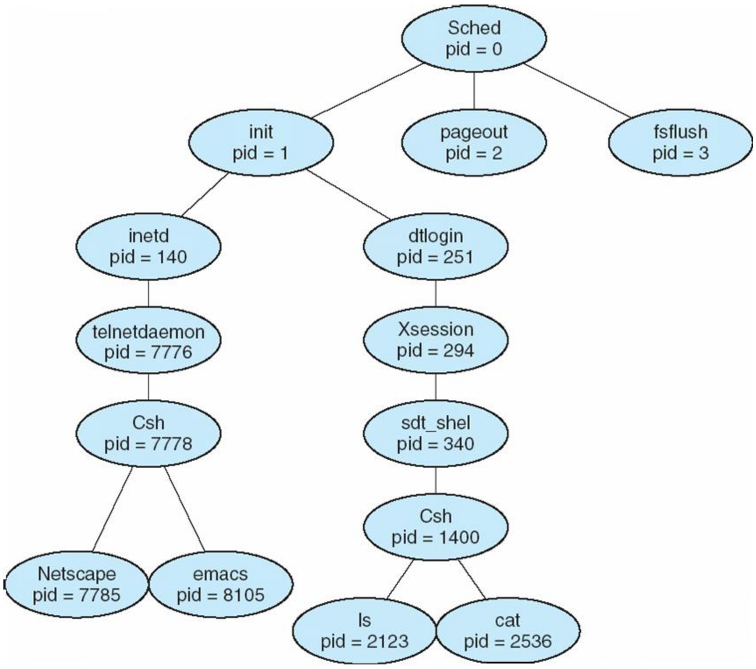
شكل رقم (3-1): حالات العملية.

4. 3. إنشاء العملية

عند إنشاء العملية تعرف وتدار برقم غير متكرر يسمى رقم تعريف العملية (Process Identification Number (PID)).

هنالك أسباب مختلفة لإنشاء العملية مثل:

- تهيئة النظام: عند إقلاع نظام التشغيل تنشأ العديد من العمليات، منها ما يعمل في الخفاء (background)، ومنها ما يعمل ويتخاطب مع المستخدم.
- عملية تستدعى النظام لإنشاء عملية أخرى: أحيانا تقوم عملية منفذة بتشغيل (إنشاء) عملية أخرى تساعدها في عملها، تعتبر العملية الأولى أب (parent) للعملية الثانية والتي تعتبر ابن (child)، العملية الابن يمكنها إنشاء عمليات أبناء لها مما قد يكون شجرة من العمليات (الشكل 3-2). قد تعمل هذه العمليات معا في وقت واحد أو تنتظر بعضها البعض.
- طلب المستخدم إنشاء عملية جديدة: عند ما ينقر المستخدم نقرا مزدوجا على أيقونة برنامج فهذا طلب من المستخدم لإنشاء عملية جديدة.
- المهام المحزمة (batch): هنا يضع المستخدم حزمة من العمليات ويطلب من نظام التشغيل تنفيذها، فيقوم النظام بتنفيذ العملية الأولى في الحزمة، ثم متى ما أتيحت له موارد العملية الثانية سيقوم بإنشائها وتنفيذها وهكذا إلى أن ينفذ كل العمليات الموجودة بالحزمة.



شكل رقم (3-2): شجرة عمليات.

3.4.1 إنشاء عملية جديدة على لينكس (fork())

يمكن استخدام استدعاء النظام fork() لإنشاء عملية جديدة، وهي لا تحتاج مدخلات (agrements)، وعند استدعاءها ترجع لنا رقم تعريف العملية التي أنشأتها (process ID). إذن هدف fork() هو إنشاء عملية جديدة تكون أبن للعملية التي استدعتها. بعد أن يتم إنشاء العملية الأبن، تنفذ العملية الإبن والعملية الأب الأمر الذي يلي fork(). لذلك لابد من التمييز بين العملية الإبن والعملية الأب وذلك بإختبار القيمة الراجعة من fork():

- فإذا كانت القيمة الراجعة من fork() سالبة، فهذا يعني أن إنشاء العملية قد فشل.
- إذا أرجعت الدالة fork() صفر للعملية الإبن، فهذا يعني أن العملية قد تم إنشاءها بنجاح.

- ترجع fork() رقم موجب للعملية الأب التي استدعتها يمثل رقم العملية (process ID).
- رقم العملية هو متغير من النوع pid_t المعرف في sys/types.h. ويمكن تمثيل رقم العملية برقم، ويمكننا استخدام الأمر getpid() للحصول على رقم العملية. البرنامج التالي يوضح كيفية استخدام fork()، حيث قمنا بإنشاء عملية بإستدعاء fork()، ثم حصلنا على رقم تعريف العملية بالامر getpid()، أمر printf سينفذ مرة بواسطة العملية الأب ومرة بواسطة العملية الإبن، مخرجا قيمتين مختلفتين للمتغير pid. يمكن اختصار خطوتي إنشاء العملية والحصول على رقم العملية (fork(); pid=getpid()) في أمر واحد هو pid=fork().

```
#include <stdio.h>

#include <sys/types.h>

void main(void) {

        pid_t pid;

        fork();

        pid = getpid();

        printf("This line is from pid %d, value = %d\n", pid, i);

}
```

يمكن استخدام استدعاءت نظام أخرى في لينكس مثل استدعاء النظام (exec) للتنفيذ، exit لإنهاء العملية.

3.5. إنهاء العملية

بعد تنفيذ العملية لآخر أمر فيها ستطلب من نظام التشغيل أن يقوم بحذفها وذلك بإستدعاء نداء النظام exit مثلا. مخرجات العملية المحذوفة ترسل

للعملية الأب عبر استدعاء النظام wait، بينما يقوم نظام التشغيل بتحرير كل موارد العملية .

قد يقوم الأب بإنهاء العملية الابن (abort) منها:

- زاد الابن في عدد الموارد المخصصة له.
- لم يعد الأب يحتاج لما يقوم به الابن (المهمة المنفذة لم نعد بحاجة لها).
- إذا أنهى الأب عمله exiting.
- بعض نظم التشغيل لا تسمح للأبن بمواصلة التنفيذ إذا أنهى الأب عمله.
- كل الأبناء سينتهون بانتهاء الأب cascading termination.

أسباب إنتهاء العملية:

- انتهاء طبيعي (إكتمل عملها).
- الانتهاء بسبب حدوث خطأ.
- تم إنهاءها بعملية أخرى .

يمكن تطبيق استدعاء ات النظام التي تنشئ أو تنهي عمليات على نظام التشغيل لينكس من داخل برنامج C. أمثلة لاستدعاءات نظام موجودة في لينكس:

- fork() لإنشاء عملية جديدة.
- exec() لتنفيذ عملية.
- exit() لإنهاء عملية والخروج.
- wait() للإنتظار.
- Kill() لإنهاء عملية.

توجد بعض البرامج المكتوبة بلغة C والتي تستخدم هذه الإستدعاءات بالملاحق. وتوجد خطوات تنفيذها على نظام التشغيل أوبونتو (يمكن تطبيقها على أي توزيعة لينكس آخرى)، أرجع للملاحق.

3.6. مثال تشبيهي

افترض أن هنالك مكتب ما، يقدم خدمات للعملاء. إذا كان هنالك موظف واحد بالمكتب (وهذا يشبه الحاسب ذو المعالج الواحد (single processor system)). فإن كل عميل يحضر للمكتب لإنجاز معاملة سيذهب لهذا الموظف، وقد يجد قبله عملاء قد حضروا مسبقا لإنجاز معاملاتهم (يجد صف انتظار)، فيضطر أن يقف في آخر الصف. في هذه الحالة سيكون كل العملاء المنتظرين في الصف في حالة جاهزية (ready)، وسيكون هنالك عميل واحد فقط (أول من وصل) يخدم بواسطة موظف المكتب (هذا العميل نقول أنه في حالة تنفيذ (running))، وقد تنجز معاملته ويخرج من المكتب حامدا ربه وشاكرا (وهنا نقول أن هذا العميل قد اكتمل عمله (terminated). ولكن ماذا يحدث إذا وجد الموظف المسئول أن أوراق العميل غير مكتملة مثلا ؟ سيوقف الموظف إنجاز معاملة هذا العميل لحين إكمال أوراقة (يحجزه (blocked)). هنا على العميل أن يخرج من المكتب ويذهب ليحضر الأوراق الناقصة. بعد عناء ومشقة تحصل صاحبنا أخيرا على بقية الأوراق المطلوبة ورجع إلى المكتب لإكمال معاملته (الآن تحول من حالة الحجز (نقص الأوراق) إلى حالة الجاهزية (الحصول على الأوراق المطلوبة)). وسينتظر دوره مرة أخرى ليسمح له الموظف المسئول (المعالج) بإكمال معاملته (التنفيذ).

عندما يفرغ الموظف من معاملة العميل الذي أمامه، سيصبح جاهزاً لاستقبال عميل جديد (غالبا العميل الذي يكون في بداية الصف)، هذا العميل الجديد سيتحول الآن من الجاهزية إلى التنفيذ، وهكذا يسير النظام.

إذا كان لدينا في المكتب أكثر من موظف فسيتم خدمة أكثر من عميل في نفس الوقت (يشبه تعدد المعالجات في نظام الحاسب).

3.7. معلومات العملية *Process Control Blocks (PCB)*

لكل عملية بنية بيانات (data structure) تسمى (PCB) تخزن فيها المعلومات الأساسية للعملية، شكل رقم (3-4)، تجمع البنيات الأساسية لكل العمليات في جدول يسمى جدول العمليات (process table)، حيث يكون هنالك خانة لكل بنية عملية بالنظام. تحتوي بينة العملية على معلومات عن العملية مثل:

- رقم تعريف العملية (process identification).
- حالة العملية (process state).
- محتوى عداد البرامج (Program counter).
- مسجلات المعالج CPU registers.
- معلومات جدولة المعالج CPU scheduling information.
- معلومات إدارة الذاكرة Memory-management information.
- معلومات الحسابات Accounting information.
- معلومات حالات الدخل والخرج I/O status information.
- مقدار ما نفذ من العملية.
- مكان الذاكرة المستخدم من قبل العملية.
- الموارد التي تستخدمها العملية مثل الملفات المفتوحة بواسطة العملية.
- أولية العملية.

حالة العملية	مؤشر
رقم العملية	
عداد البرامج	
محتوى المسجلات	
حدود الذاكرة	

الملفات المفتوحة
.....

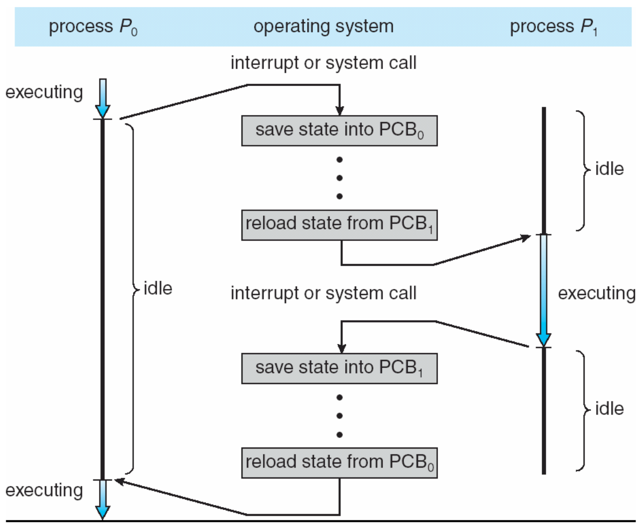
شكل رقم (3-4): بينية معلومات العملية (PCB).

3.8 التحول السياقي (*Context switch*)

تستخدم معلومات العمليات عندما تتحول العملية من حالة التنفيذ إلى حالة الجاهزية أو الحجز. عندها يقوم المعالج بتوقيف عملية وتنفيذ عملية أخرى، حيث يتم حفظ معلومات العملية التي تم توقيفها (مثلا العملية P0) في PCB0، ثم يتم تحميل معلومات العملية المراد تنفيذها (مثلا P1) من PCB1.

إذا أراد نظام التشغيل تنفيذ P0 مرة أخرى فسيقوم بتخزين معلومات P1 في PCB1 ثم تحميل معلومات P0 من PCB0 حيث يستطيع مواصلة التنفيذ من آخر نقطة وقفت فيها العملية P0 (الشكل 3-5).

الزمن المستغرق في الإنتقال بين عمليتين يكون مهدور وغير مستفاد منه، حيث لا يقوم النظام بعمل في هذه الفترة.

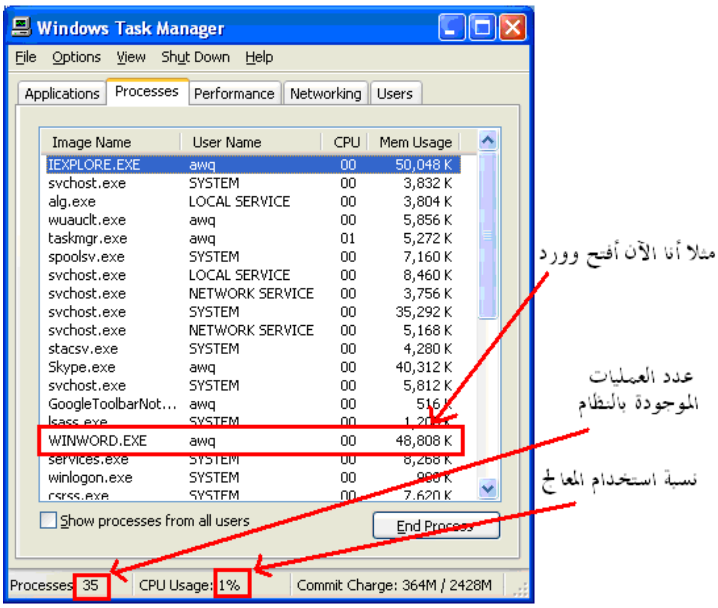


شكل رقم (3-5): إنتقال المعالج بين عمليتين (P0 و P1).

3.9 العمليات في ويندوز

عندما ننفذ برامجنا على ويندوز، لابد لنظام التشغيل من معرفة كيف يدير هذه البرامج للتأكد من أن كل برنامج أخذ حصته من الوقت في المعالج وفي الوصول إلى الذاكرة وأجهزة الدخل والخرج. ولتحقيق ذلك يتعامل نظام التشغيل مع كل برنامج كعملية. فإذا قمنا بتشغيل برنامج فسينشئ عملية لهذا البرنامج، وإذا نفذنا نسخة من نفس البرنامج فسيقوم نظام التشغيل بإنشاء عملية أخرى لهذه النسخة، بحيث تكون هنالك عملية لكل نسخة شغالة من البرنامج. وبالتالي كل البرامج التي تعمل في نظامك هي عبارة عن عمليات يدير ويتابع عملها نظام التشغيل.

لمعرفة العمليات التي تنفذ بجهازك حاليا قم بالضغط على Ctrl+Alt+Del فتظهر نافذة، أنقر على تبويب Processes فترى كل العمليات التي تعمل الآن في جهازك بما فيها عمليات نظام التشغيل ومضادات الفيروسات والبرامج الخدمية وكل برامجك المفتوحة، وترى حجم الذاكرة الذي تستخدمه كل عملية، الشكل (3-6).



شكل رقم (3-6): مشاهدة العمليات في ويندوز.

من الشكل (3-6) نجد أن كل عملية لديها مالك (user name) هو الذي شغلها. أيضا يظهر تحت CPU، زمن المعالج الحالي المستخدم لكل عملية، كذلك المساحة

المستخدمة من الذاكرة والتي تخزن فيها العملية شفراتها وبياناتها (Mem Usage). مثلا برنامج وورد (WINWORD.EXE) هو عملية قام بتشغيلها المستخدم awq، وتستخدم ذاكرة حجمها 48,808k.

يمكنك النقر على أي عملية وإيقافها من مدير المهام بالنقر على العملية ثم النقر على الزر End Process. مع التنبيه إلى أن كل مستخدم يستطيع أيقاف عملياته، إذا كان مستخدم عادي، ولا يستطيع إيقاف عمليات المستخدمين الآخرين إلا إذا كان مشرف Administrator.

بالرغم من أن ويندوز تبدو لك أنها تشغل أكثر من برنامج (عملية) في نفس الوقت، إلا أن هذا غير صحيح للأجهزة ذات المعالج الواحد. فالحقيقة أن عملية واحدة فقط تعمل في الوقت الواحد، ثم تخرج عندما تكون عاطلة (idle)، وتنفذ عملية غيرها. أما في الحاسبات التي تمتلك أكثر من معالج فيمكن تشغيل أكثر من عملية، بحيث تشتغل كل عملية في معالج وفي نفس الوقت.

استخدام مكتبة .NET للتعامل مع العمليات

توجد صفوف في مكتبة .NET تسمح لبرامجك بالوصول لبيانات العمليات التي تعمل في جهازك. ويمكنك استخدام هذه المعلومات لمعرفة حالة برنامجك الحالي أو للحصول على معلومات عن بقية العمليات التي تعمل الآن في جهازك.

معرفة معلومات برنامجك الحالي باستخدام visual basic.net

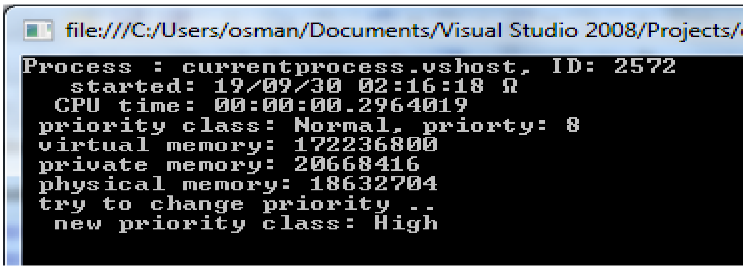
البرنامج التالي يستخدم الصف process الموجود في مكتبة .NET لمعرفة معلومات العملية الحالية (هذا البرنامج). الشفرة التالية توضح ذلك:

```
Imports System
Imports System.Diagnostics
Module Module1

    Sub Main()
       Dim thisProcess As Process
       thisProcess = Process.GetCurrentProcess
       Dim pname As String = thisProcess.ProcessName
       Dim started As DateTime = thisProcess.StartTime
       Dim pID As Integer = thisProcess.Id
       Dim mem As Integer = thisProcess.VirtualMemorySize64
       Dim phymem As Integer = thisProcess.WorkingSet64
```

```
        Dim primem As Integer = thisProcess.PrivateMemorySize64
        Dim priority As Integer = thisProcess.BasePriority
        Dim priClass As ProcessPriorityClass =
             thisProcess.PriorityClass
         Dim cpuTime As TimeSpan =
             thisProcess.TotalProcessorTime
         Console.WriteLine("Process : {0}, ID: {1}", pname,pID)
         Console.WriteLine("   started: {0}", started.ToString)
         Console.WriteLine("  CPU time: {0}", cpuTime.ToString)
         Console.WriteLine(" priority class: {0}, priorty:
                             {1}", priClass, priority)
         Console.WriteLine(" virtual memory: {0}", mem)
         Console.WriteLine(" private memory: {0}", primem)
         Console.WriteLine(" physical memory: {0}", phymem)
         Console.WriteLine(" try to change priority ..")
         thisProcess.PriorityClass = ProcessPriorityClass.High
         priClass = thisProcess.PriorityClass
         Console.WriteLine("  new priority class: {0}",
                             priClass)
         Console.Read()
     End Sub
End Module
```

عند تنفيذ البرنامج أعلاه (يصبح عملية) وتظهر لنا معلوماته، مثل رقم العملية ID، ما يتسخدم من ذاكرة حقيقية وذاكرة ظاهرية ومعالج. كذلك يظهر البرنامج أولوية العملية (priority) مع إمكانية تغيير هذه الأولوية.



أيضا يمكننا معرفة المعلومات عن جميع العمليات بإنشاء مصفوفة تخزن معلومات كل العمليات بالشفرة التالية:

```
Dim allProc() as Process

allProc = Process.GetProcesses()
```

ثم نستخدم التكرار لعرض معلومات كل عملية كما يلي:

Foreach thisProc as Process in allProc

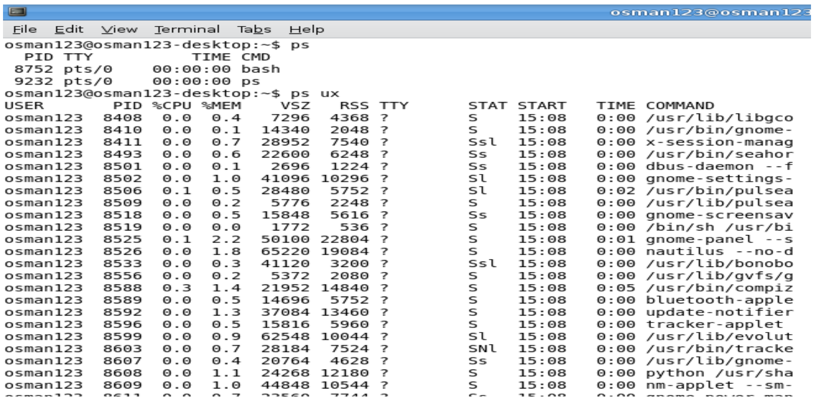
أعرض معلومات العملية

Next

9.10. العمليات في لينكس

يمكنك في لينكس معرفة العمليات التي تعمل الآن في جهازك، بما فيها رقم تعريف العملية (process identification number (PID))، باستخدام الأمر ps.

يمكننا استخدام الأمر ps في نظام التشغيل أوبونتو (Ubuntu)، وهو أحد توزيعات لينكس، لمشاهدة العمليات التي تعمل الآن في النظام كما في الشكل (3-7).



شكل رقم (3-7): مشاهدة العمليات في لينكس (أوبونتو) بالأمر: ps us.

أيضا هنالك العديد من استدعاءات النظام التي تمكننا من التعامل مع العمليات والتي ذكرناها في 3.4.1.، كذلك توجد الكثير من الامثلة البرمجية بالملحق (د).

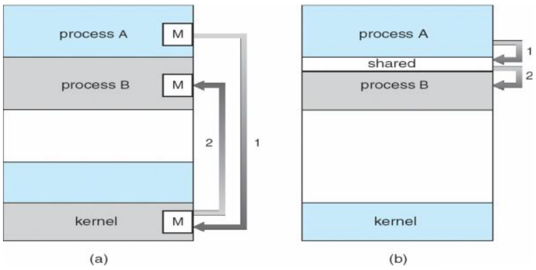
لكل عملية رقم مفرد يسمى PID . وقد نطلق أحيانا على العملية كلمة "مهمة" (task)، لذلك نجد "مدير المهام" (task manager) في ويندوز.

3.11 الاتصال بين العمليات

قد تكون العمليات الموجودة في النظام مستقلة أو متعاونة (independent or cooperating) . العمليات المتعاونة تؤثر وتتأثر بما حولها من عمليات، أما العمليات المستقلة فلا.

العمليات المتعاونة تحتاج اتصال فيما بينها interprocess communication(IPC)، حيث يوجد نوعين من طرق الاتصال، كما موضح في الشكل (3-8)، هي:

- وجود ذاكرة مشتركة Shared memory
- عن طريق تبادل الرسائل Message passing



شكل رقم (3-8): الاتصال بين العمليات.

3.12. تمارين محلولة

أختار الإجابة الصحيحة:

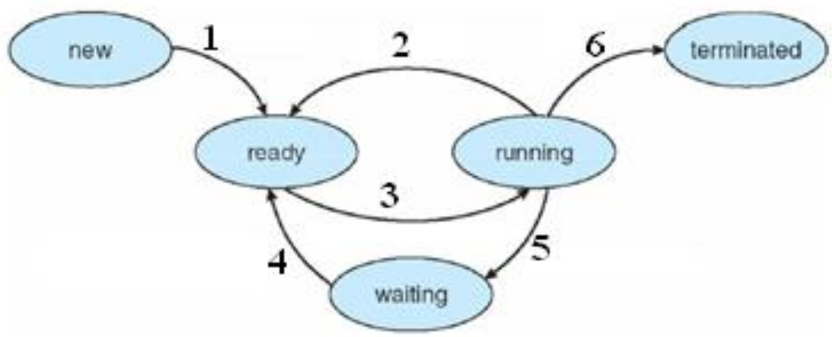
1. العمليات المتعاونة تتصل فيما بينها للآتي:

- لنشر المعلومات.
- لتقليل سرعة التنفيذ.
- للبعد الجغرافي.
- لا شيء مما ذكر (1)

2. أجب بنعم أو لا مع تصحيح الإجابة الخاطئة:

- عندما ينتقل المعالج لتنفيذ عملية، على النظام حفظ PCB العملية القديمة وتحميل PCB العملية الجدية. نعم
- قد يقوم الأب بإنهاء العملية الابن (abort) إذا زاد الأب في عدد الموارد المخصصة له. لا، إذا زاد الأبن
- نستخدم في لينكس استدعاء النظام (exec system call) لإنشاء عملية جديدة. لا، fork()
- عندما تنشئ عملية عملية أخرى سيكون تنفذ العملية وأبناءها بالتوالي فقط. لا، قد يكون بالتوالي أيضا.

3. وضح أسباب الانتقال بين حالات العمليات المبنية بالرسم أعلاه:



1 . دخول 2. مقاطعة 3. مجدول 4. إكتمال حدث/دخل/خرج

5. .حدث/دخل/خرج 6. خروج

3.13. تمارين غير محلولة

1. عرف العملية (process) ؟

2. أذكر حالات العملية (process states) مع توضيح العلاقة بينهم بالرسم ؟

3. ما هي محتويات بنية معلومات العملية (PCB) ؟

4. ما هي مسببات انشاء العملية ؟

5. أذكر مسببات إنهاء العملية ؟

6. أجب بنعم أو لا (مع تصحيح الإجابة الخاطئة)

- لتنفيذ البرنامج يجب تحميله من الرام إلى القرص الصلب أو لا؟
- عندما ينتقل المعالج لتنفيذ عملية، على النظام حفظ معلومات العملية التي نريد توقيفها (PCB) وتحميل PCB العملية الجديدة، عبر ما يسمى المشاركة.
- قد يقوم الأب بإنهاء العملية الابن (abort) إذا زاد الأب في عدد الموارد المخصصة له.
- نستخدم في لينكس استدعاء النظام (exec system call) لإنشاء عملية جديدة.

7. عندما تنشئ عملية عملية أخرى سيكون تنفذ العملية وأبنها بالتوالي أم بالتوازي ؟

8. اذكر تنفيذ البرنامج (من تحميله في الذاكرة إلى إكتماله)؟

الباب الرابع:

جدولة المعالج

الباب الرابع

جدولة المعالج (CPU Scheduling)

تعدد البرمجة (multiprogramming) معناها أن هنالك أكثر من عملية جاهزة (بالذاكرة الرئيسية). مهمة اختيار عملية من العمليات الجاهزة لتنفذ في المعالج هي مهمة المجدول (scheduler)، والطريقة (الخوارزمية) التي يستخدمها المجدول لانتقاء العملية تسمى خوارزمية الجدولة (scheduling algorithm).

الغرض من جدولة العمليات هي اختيار عملية من العمليات الموجودة بالذاكرة لتنفذ في المعالج، بحيث يكون المعالج دوما مشغولا (كلما زاد انشغال المعالج كلما زادت كفاءته CPU utilization).

4.1. دورة حياة العملية (*CPU I/O Burst Cycle*)

الهدف من تعدد البرمجة (multiprogramming) هو أن تكون هنالك دائما عملية تحت التنفيذ بحيث نستفيد إستفادة قصوى من المعالج.

في الحاسب ذو المعالج الواحد ستكون هنالك عملية واحدة فقط تعمل داخل المعالج في اللحظة الواحدة. ويستفاد من تعدد البرمجة (multiprogramming) في أنه إذا احتاجت عملية داخل المعالج إلى إجراء دخل أو خرج (وهذا يستغرق وقت كبير جدا مقارنة مع زمن المعالجة)، سيكون المعالج عاطل لا يعمل في انتظار الدخل أو الخرج، لذلك يتم اخراج العملية التي بالمعالج وإدخال أخرى ليستفاد من المعالج وجعله مشغولا دوماً. عند إكتمال الدخل أو الخرج تستطيع العملية التي تم إخراجها مواصلة التنفيذ (عندما تجد فرصة في المعالج).

إذن العملية تكون في المعالج شغالة ثم قد يتم توقفها عندما تحتاج دخل أوخرج ثم ترجع لتعمل بعد إكتمال الدخل أو الخرج، وقد تتوقف مرة أخرى لدخل أو خرج ثم ترجع لتعمل مرة أخرى وهكذا قد تتنقل العملية بين دخل وخرج و معالجة حتى تنتهي. أي تقضي العملية وقتها بين تنفيذ داخل المعالج (CPU burst) وإنتظار لدخل أو خرج (I/O burst). إذا كانت الفترات التي تقضيها العملية داخل المعالج أطول من التي تقضيها في إنتظار دخل أو خرج فنقول أن العملية (CPU bound)، أما إذا كان وقتها الذي تقضيه في إنتظار الدخل والخرج أكبر من الذي تقضيه داخل المعالج فنقول أن

العملية (I/O bound). يعتمد هذا على نوع العملية فهنالك عمليات تحتاج إجراء حوسبة كثيرة ولا تحتاج ولا تظهر معلومات كثيرة، فتكون هذه من النوع CPU bound، أما العمليات التي تحتاج إلى دخل وخرج كثير، مثل إدخال البيانات وتخزينها فستكون من النوع I/O bound.

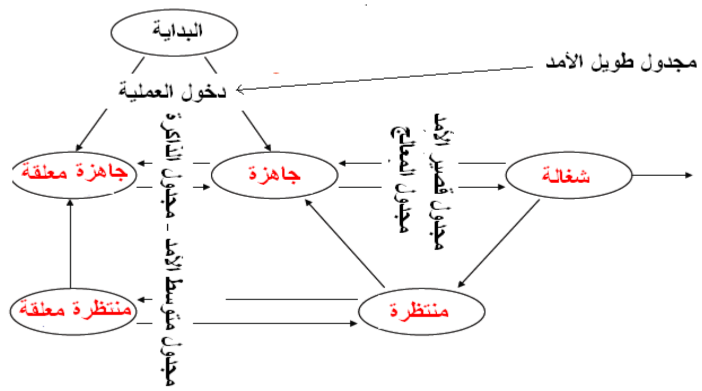
4.2 أنواع المجدول

مهمة المجدول هو تنظيم دخول العمليات إلى المعالج. هنالك ثلاث أنواع من المجدولات، ولكل مجدول عمل يقوم به، هي:

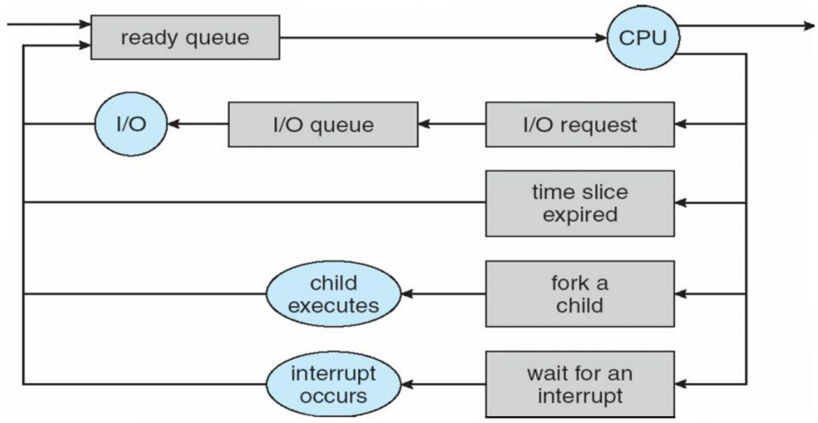
- مجدول المهام أو المجدول طويل الأمد long-term scheduler (or job scheduler) والذي يقوم باختيار العمليات من القرص وتحميلها في الذاكرة لتكون جاهزة للتنفيذ.

- المجدول متوسط الأمد medium-term scheduler.

- مجدول المعالج short-term schedulerأو المجدول قصير الأمد (or CPU scheduler) الذي يقوم بانتقاء عملية من العمليات الجاهزة الموجودة بالذاكرة ويحجز لها المعالج.الفرق بين الاثنين في سرعة اختيار العملية ، فبينما الأول يكون بطيئا في إحضار العمليات من القرص إلى الذاكرة يكون الأول سريع في إدخال العمليات من الذاكرة إلى المعالج، الشكل (4-1).



شكل رقم (4-1): أنواع المجدولات



شكل رقم (4-2): مجدول المعالج

4.2.1. مجدول العمل Long term scheduler – job scheduler

- اختيار العمليات التي ستحمل بالذاكرة (إدخال العمليات إلى صف الانتظار).
- يحدد أي عملية ستبدأ اعتمادا على الترتيب والأفضلية.
- لا يستخدم في أنظمة التقسيم الزمني (time sharing systems).

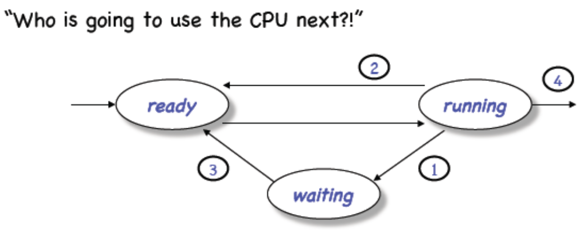
4.2.2. المجدول المتوسط (Medium term scheduler)

- يجدول العمليات بناءا على الموارد التي تحتاج (ذاكرة/ دخل/خرج).
- يقوم بتعليق (suspend) العمليات التي لم تتوفر لها مواردها حاليا.
- عادة تكون الذاكرة مورد محدود وهنا يعمل مدير الذاكرة كمجدول متوسط الأمد (استخدام الذاكرة الظاهرية).

4.2.3. مجدول المعالج (قصير الأمد)

- يقوم المجدول باختيار عملية من العمليات الموجودة بالذاكرة والجاهزة للتنفيذ (ready queue) وحجز المعالج لها.

- تحديد العملية التالية التي ستستخدم المعالج بعد فراغه من العملية الحالية (الشكل 4-2).
- يجب أن يكون المجدول سريع جدا.
- إذا احتاجت عملية مورد أو دخل/خرج ستزال من المعالج وتحول إلى حالة الانتظار وإدخال عملية أخرى من صف الانتظار إلى المعالج.
- يتخذ المجدول قراراته عند ما تتحول العملية من:
    1. من شغالة إلى منتظرة
    2. من شغالة إلى جاهزة
    3. من منتظرة إلى جاهزة
    4. انتهت Terminates
- في الحالات 1 و4 ستخرج العملية إجباريا إما لإنتهاءها أو لأنها تحتاج دخل أو خرج، في هذه الحالة لابد للمجدول من إختيار عملية بديلة، فليس لديه خيار آخر. في هذه الحال نقول أن المجدول non-preemptive.
- أما 2،4 فإن العملية تخرج إختيارا، وقد تواصل، هنا للمجدول حق الإختيار في إخراجها أو لا. هنا نقول أن المجدول preemptive.



الجدولة غير قابلة للتوقف (non-preemptive scheduling) : عند ما تدخل عملية المعالج وتبدأ التنفيذ فإنها لن تترك المعالج أبدا إلا في إحدى حالتين :

- الاكتمال.

- التحول إلى حالة الانتظار.

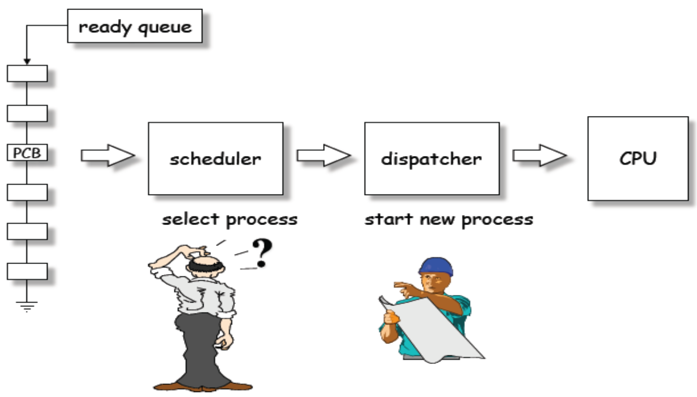
4.2.3.1. المرسل Dispatcher

المرسل هو جزء من المجدول يقوم بإرسال العمليات التي يختارها مجدول المعالج، ومن وظائفه:

- التحول السياقي switching context: حفظ بيانات العملية الموقفة وتحميل بيانات العملية التي تم اختيارها لتعمل في المعالج.
- التحول إلى وضع المستخدم user mode.
- القفز إلى الموقع المناسب في برنامج المستخدم لإعادة تشغيل البرنامج.

المرسل لابد من أن يكون سريع جدا، لأنه سيعمل مع كل عملية.

زمن الإرسال Dispatch latency: هو الوقت الذي يستغرقه المرسل في توقيف عملية وتشغيل أخرى (تحويل عملية).



4.2. معايير الجدولة (*Scheduling Criteria*)

تستخدم بعض المصطلحات والقياسات التي تمكننا من المقارنة بين طرق الجدولة المختلفة، مثل:

- استغلال المعالج (CPU utilization): جعل المعالج مشغولا بقدر ما نستطيع.
- الإنتاجية (Throughput): عدد الـعمليات التي يمكن إنجازها في فترة زمنية معينة.
- الزمن الإكتمال (الزمن الكلي) (Turnaround time): الزمن المستغرق لتنفيذ عملية ما (من دخولها (إنشاءها) حتى انتهاءها).
- زمن الانتظار (Waiting time): الزمن الذي تقضيه العملية في صف الانتظار (ready queue)، أي مجموع فترات الانتظار التي تقضيها العملية في صف الانتظار (ready queue) أثناء تنفيذها.
- زمن الاستجابة (Response time): الزمن المستغرق من دخول العملية (new) إلى ظهور أول مخرج (استجابة) من العملية.

4.4. تحسين الأداء

يسعى نظام التشغيل إلى تحسين أداء المعالج بتحقيق الآتي:

- زيادة استغلال للمعالج: زيادة استغلال المعالج بأقصى ما يمكن.
- زيادة الانتاجية: زيادة عدد العمليات المنفذة في فترة زمنية معينة.
- تقليل الزمن الكلي: تقليل الزمن الكلي لتنفيذ العمليات بقدر ما يمكن.
- تقليل زمن انتظار : تقليل زمن الانتظار لكل عملية بقدر ما يمكن.
- تقليل زمن الاستجابة (response time): جعل العمليات تستجيب وتبدأ إظهار مخرجاتها بسرعة.

4.5. خوارزميات الجدولة (***Scheduling Algorithms***)

هنالك العديد من خوارزميات الجدولة مثل:

4.5.1. القادم أولا يخدم أولا (First-Come, First served)

- تنفذ العمليات ترتيب وصولها واسطة مجدول المعالج (قصير الأمد)، حيث تنفذ أول عملية وصلت إلى صف الانتظار أولاً ثم التي وصلت بعدها ثانيا، وهكذا. وهي خوارزمية بسيطة في عملها وواضحة.
- يحذف المجدول العملية من المعالج إذا أرادت التحول إلى وضع الانتظار أو انتهت.
- الخوارزمية مناسبة للعمليات الكبيرة عندما تجد فرصة للتنفيذ، وتسبب مشكلة للعمليات القصيرة إذا كانت خلف عمليات كبيرة

مثال (1)

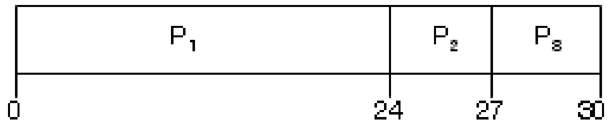
الجدول التالي يوضح عمليات في صف الانتظار ونريد تنفيذها حسب خوارزمية القادم أولا يخدم أولا. حيث يمثل عمود العملية اسم العملية والعمود الثاني يمثل الوقت الذي تحتاجه كل عملية لتكمل عملها داخل المعالج، أحسب الآتي :

1. متوسط زمن الانتظار (waiting time).
2. متوسط زمن الاكتمال (completion time).
3. أحسب متوسط زمن الانتظار إذا كان ترتيب الوصول عكسي (p3 ثم p2 ثم p1) ثم وصح الفرق بين المتوسطين.

العملية	الزمن
P1	24
P2	3
P3	3

الحل

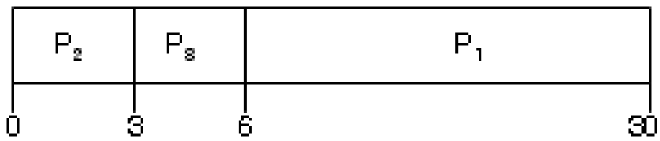
إذا وصلت العمليات بالترتيب P1 ثم P2 ثم P3، وتم خدمتها بترتيب القادم أولا يخدم أولا، فإن المجدول سينفذ العمليات على المعالج حسب ترتيبها كالرسم التالي : (Gantt Chart)



1. متوسط زمن الانتظار = (0+24+27)/3 = 17ملي ثانية.

2. متوسط زمن الاكتمال = (24+27+30)/3 = 27 ملي ثانية.

3. إذا وصلت العمليات بالترتيب العكسي فإن شكل العمليات داخل المعالج سيكون كالآتي:



عليه فإن متوسط زمن الانتظار = (0+3+6)/3 = 3 ملي ثانية.

ومتوسط زمن الاكتمال سيكون = (3+6+30)/3 = 13 ملي ثانية.

حيث نلاحظ أن متوسط زمن الانتظار إذا نفذت العمليات الكبيرة في النهاية أفضل من متوسط زمن الانتظار إذا تم تنفيذ العمليات الكبيرة أولا.

4.5.2. العملية الأقصر أولا **Shortest-Job-First (SJF)**

- إفتراض أننا نعلم مقدما كم ستحتاج كل عملية بصف الانتظار من وقت داخل المعالج، فإن المجدول سيختار العملية التي تحتاج زمن أقل أولا.
- تعتبر هذه الخوارزمية مثالية حيث إنها تعطي أقل متوسط زمن انتظار.

- المشكلة الأساسية في هذه الخوارزمية هي معرفة طول الوقت الذي ستحتاجه العملية داخل المعالج مسبقا، فغالبا لا يعلم المجدول ذلك مما يصعب معرفة العملية الأقصر.
- سيكون الأداء ضعيف في حالة وصول عمليات قصيرة بعد بدء تنفيذ عملية طويلة.
- كذلك قد يحدث حرمان للعمليات الطويلة (تحرم من التنفيذ لوجود عمليات قصيرة).
- يقدم خدمة جيدة للعمليات القصيرة.
- تفشل الخوارزمية مع العمليات التي كانت سابقا قصيرة لكنها الآن أصبحت طويلة.

تنقسم الخوارزمية إلى نوعين هما :

- غير قابلة للتوقف (Non-preemptive): إذا بدأت عملية التنفيذ داخل المعالج لن تتوقف لعملية أخرى.
- قابلة للتوقف (Preemptive): إذا وصلت عملية جديدة إلى صف الانتظار وكان زمنها أقصر من الزمن المتبقي للعملية التي تنفذ حاليا بالمعالج سيقوم المجدول بإخراج الأولى وإدخال التي وصلت حديثا.

مثال (2)

مثال على العمليات الغير قابلة للتوقيف والتي وصلت في نفس الوقت (أي دون إعتار لزمن الوصول).

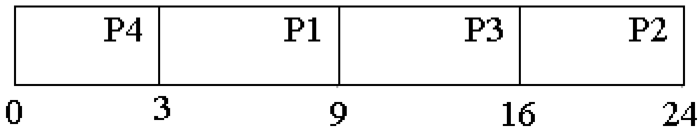
الزمن	العملية
6	P1

P2	8
P3	7
P4	3

جدول هذه العمليات باستخدام خوارزمية الأقصر أولا ؟

الحل

سنقوم بعمل رسم شكل (Gantt chart) يوضح كيف ستكون العمليات داخل المعالج:



متوسط زمن الانتظار = (0+3+9+16)/4 = 7 ملي ثانية.

متوسط زمن الاكتمال = (3+9+16+24)/4 = 13 ملي ثانية.

مثال (3)

مثال على عمليات لم تصل في وقت واحد باستخدام خوارزمية الأقصر أولا القابلة للتوقف.

العملية	زمن الوصول	الزمن
P1	0	8
P2	1	4
P3	2	9
P4	3	5

إذا كانت العمليات إلى صف الانتظار حسب عمود زمن الوصول المبين أمام كل عملية بالجدول فوضح كيف يستخدم المجدول خوارزمية الأقصر أولا القابلة للتوقيف (preemptive).

الحل

سيكون ترتيب العمليات داخل المعالج كما يلي:

P1	P2	P4	P1	P3
0 – 1	1 – 5	5 – 10	10 – 17	17 – 26

زمن انتظار العملية = زمن الانتظار الكلي – زمن الوصول، بالتالي فإن:

زمن انتظار P1 = (0 + 10) - 0 = 10

زمن انتظار P2 = 1 – 1 = 0

زمن انتظار P3 = 17- 2 = 15

زمن انتظار P4 = 5- 3 = 2

متوسط زمن الانتظار = (10+0+15+2)/4 = 6.75 ملي ثانية.

متوسط زمن الاكتمال = (17+ 5+ 26+ 10)/4= 14.5 ملي ثانية.

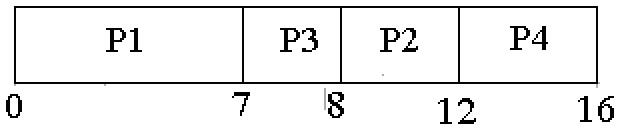
مثال (4)

مثال آخر على عمليات لم تصل في وقت واحد باستخدام خوارزمية الأقصر أولا الغير قابلة للتوقف.

العملية	زمن الوصول	زمن العملية
P1	0	7
P2	2	4
P3	4	1
P4	5	4

الحل

ترتيب العمليات داخل المعالج يكون كما يلي:



زمن انتظار P1 = 0 - 0 = 0

زمن انتظار P2 = 8 - 2 = 6

زمن انتظار P3 = 7 - 4 = 3

زمن انتظار P4 = 12 - 5 = 7

متوسط زمن الانتظار = (0+6+3+7)/4 = 4 ملي ثانية.

متوسط زمن الإكتمال = (7+8+12+16)/4= 10.75 ملي ثانية.

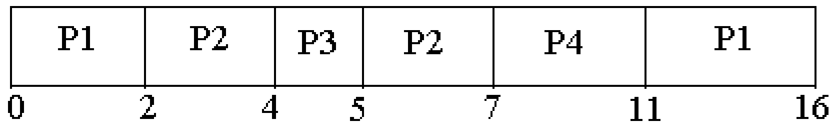
مثال (5)

هنا سنحل المثال (4) السابق بطريقة الأقصر أولا القابلة للتوقف.

العملية	زمن الوصول	زمن العملية
P1	0	7
P2	2	4
P3	4	1
P4	5	4

الحل

نرسم خارطة جانت (Gantt chart) للجدول أعلاه كما يلي:



زمن انتظار P1 = (0+(2-11)) – 0 = 9

زمن انتظار P2 = (2+(4-5)) – 2 = 1

زمن انتظار P3 = 4 – 4 = 0

زمن انتظار P4 = 7 – 5 = 2

متوسط زمن الإنتظار = (9+1+0+2)/4 = 3 ملي ثانية.

متوسط زمن الاكتمال = (16+7+5+11)/4= 9.75 ملي ثانية.

4.5.3. الأولوية (**Priority**)

كل عملية لديها رقم مرفق معها، هذا الرقم يحدد أهمية العملية، حيث سيتم تنفيذ العملية ذات الأولوية الأعلى (أقل رقم). فإذا كنا نعتبر أن الرقم الأقل يمثل الأولوية الأعلى، فهذا يعني أنه إذا كان لدي عملية برقم الأولوية 5 وهنالك عملية برقم الأولوية 7، فسينفذ المجدول العملية ذات الرقم 5 قبل العملية ذات الرقم 7، لأن أولويتها أعلى. هنالك نوعين هما:

- قابلة للتوقف (Preemptive).
- غير قابلة للتوقف (non-preemptive).

4.5.3.1. الحرمان (Starvation)

المشكلة في خوارزمية الأولوية هو الحرمان ((Starvation))، حيث لن تجد العمليات ذات الأولوية الدنيا فرصة للتنفيذ داخل المعالج طالما أن هنالك عمليات ذات أولوية أعلى منها. حيث سينفذ المعالج العمليات ذات الأولوية العليا وكلما قرب دور العمليات ذات الأولوية الدنيا للتنفيذ داخل المعالج قد تصل عمليات أخرى لها أولوية أعلى منها فتحرمها من استخدام المعالج.

حل مشكلة الحرمان هو النضوج (Aging)، أي كل ما مر زمن طويل على عملية ذات أولوية دنيا نرفع أولويتها درجة، وهكذا إذا مر وقت طويل على هذه العملية ستصبح ذات أولوية عليا ونكون بهذا مكّناها من التنفيذ.

مثال (6)

مثال على عمليات غير قابلة للتوقف (وصلت في وقت واحد).

زمن العملية	رقم الأولوية	العملية
10	3	P1
1	1	P2
2	3	P3
1	4	P4
5	2	P5

الحل

باستخدام خوارزمية الأولية سيكون شكل العمليات داخل المعالج كما يلي:

P2	P5	P1	P3	P4
0 – 1	1 – 6	6 – 16	16 – 18	18 – 19

حيث سيكون متوسط زمن الانتظار = (6+0+16+18+1)/5 = 8.2 ملي ثانية.

متوسط زمن الاكتمال = (16+1+18+19+6)/5 = 12 ملي ثانية.

مثال (7): تمرين

قم بحل المثال السابق بطريقة الأولوية الغير قابلة للتوقف بإعتبار أن العمليات لم تصل في وقت واحد (محدد زمن وصول كل عملية في عمود "زمن الوصول").

العملية	رقم الأولوية	زمن العملية	زمن الوصول
P1	3	10	0
P2	1	1	2
P3	3	2	4
P4	4	1	7
P5	2	5	9

4.5.4. التقسيم الزمني (**Round Robin (RR)**)

هذا النوع من الخوارزميات صمم خصيصا للنظم التي تستخدم المشاركة الزمنية (time sharing)، فلكل عملية حصة زمنية داخل المعالج تخرج بعدها من المعالج لتدخل العملية التي تليها فتأخذ نفس الحصة الزمنية التي أخذتها العملية الأولى، وهكذا يتم تقسيم زمن المعالج بحيث تأخذ كل عملية حصة بالمعالج، ثم تبدأ من جديد. فالخوارزمية تعمل بطريقة تشبه خوارزمية القادم أولا يخدم أولا، لكن مع إمكانية توقيف كل عملية إذا اكتملت الفترة الزمنية المقررة لها داخل المعالج. حيث تحدد

الحصة الزمنية (quantum or time slice) بقيمة صغيرة تتراوح بين 10 إلى 100 ملي ثانية.

يقوم المجدول بتقسيم زمن المعالج بين العمليات فتعطى كل عملية حصة زمنية محددة داخل المعالج، ويتم تحديد الفترة الزمنية لكل العمليات فيما يسمى Quantum، فإذا كانت قيمة Quantum هي 10، فهذا يعني أن المجدول سيسمح لكل عملية أن تنفذ في المعالج لمدة 10 ملي ثانية، ثم تخرج (ترجع إلى نهاية صف الانتظار)، وتدخل العملية التي تليها، لتأخذ 10 ملي ثانية وهكذا.

تنيهات

- إذا كان الزمن الذي تحتاجه العملية أقل من الحصة المقررة، فستأخذ العملية ما تحتاجه من الحصة وتخرج لتدخل العملية التي تليها. مثلا إذا كانت العملية تحتاج 3 ملي ثانية وكانت الحصة المقررة للعملية هي 10 ملي ثانية، فستمكث العملية بالمعالج 3 ملي ثانية وليست 10 ملي ثانية.
- لا توجد عمليات غير قابلة للتوقف في هذه الخوارزمية فكل العمليات ستجبر على التوقف عند إكتمال حصتها بالمعالج.

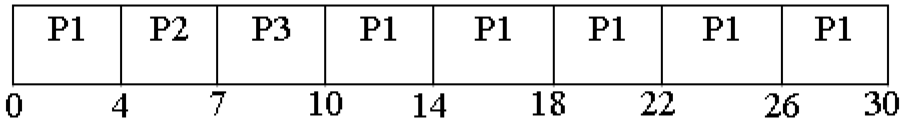
مثال (8)

في الجدول التالي عمليات حدد ت لها حصة زمنية (Quantum) تساوي 4 ملي ثانية، أحسب كل من متوسط زمن الانتظار ومتوسط زمن الاكتمال لهذا العمليات.

الزمن	العملية
24	P1
3	P2
3	P3

الحل

العمليات داخل المعالج ستكون كالشكل التالي:



زمن انتظار P1 = 0 + (10-4) = 6

زمن انتظار P2 = 4

زمن انتظار P3 = 7

متوسط زمن الانتظار = (6+4+7)/3 = 17/3 = 5.67 ملي ثانية.

متوسط زمن الاكتمال = (30+7+10)/3 = 47/3 = 15.67 ملي ثانية.

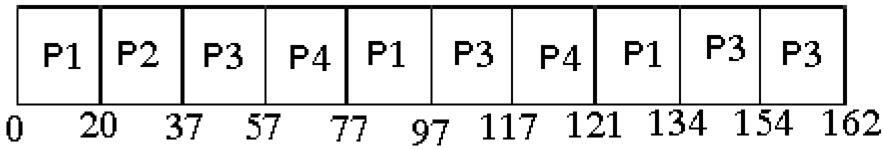
مثال (9)

إذا كانت الحصة الزمنية لكل عملية هي 20ملي ثانية، أحسب متوسط زمن الانتظار.

العملية	الزمن
P1	53
P2	17
P3	68
P4	24

الحل

الشكل التالي يوضح تنفيذ العمليات داخل المعالج:



زمن انتظار P1 = 0+(20-77)+(97-121) = 81

زمن انتظار P2 = 20

زمن انتظار P3 = 37+(57-97)+(117-134) = 94

زمن انتظار P4 = 57+(77-117) = 97

متوسط زمن الانتظار = (81+20+94+97)/4 = 73 ملي ثانية.

4.6. برنامج محاكاة خوارزميات الجدولة

يمكنك كتابة برنامج بأي لغة على فيجوال ستديو مثل C# أو VB.NET أو أي لغة أخرى مثل C,C++,JAVA لمحاكاة عمل خوارزميات الجدولة، حيث تظهر شاشة (كأدناه) تطلب من المستخدم إدخال بيانات العمليات ثم النقر على زر الخوارزمية التي نريد من البرنامج استخدامها لحساب متوسط زمن الإنتظار.



أو يمكن استخدام تطبيق كونسول (console application) في visual basic.NET لإدخال بيانات العمليات كالآتي:

```
Dim x, y As Integer
Console.Write("Enter number of processes: ")
x = Console.ReadLine()
Dim z(x - 1) As Integer
For y = 0 To x - 1
   Console.WriteLine(" Enter burst time for process " & y)
   z(y) = Console.ReadLine()
      Next
```

بعد إدخال بيانات العمليات (اسم العملية مثلا ووقتها)، سنقوم بحساب زمن الإنتظار لكل العمليات (FCFS) وذلك بالطريقة التالية:

```
wt(0) = 0
For i = 1 To x - 1
     wt(i) = z(i - 1) + wt(i - 1)
Next
```

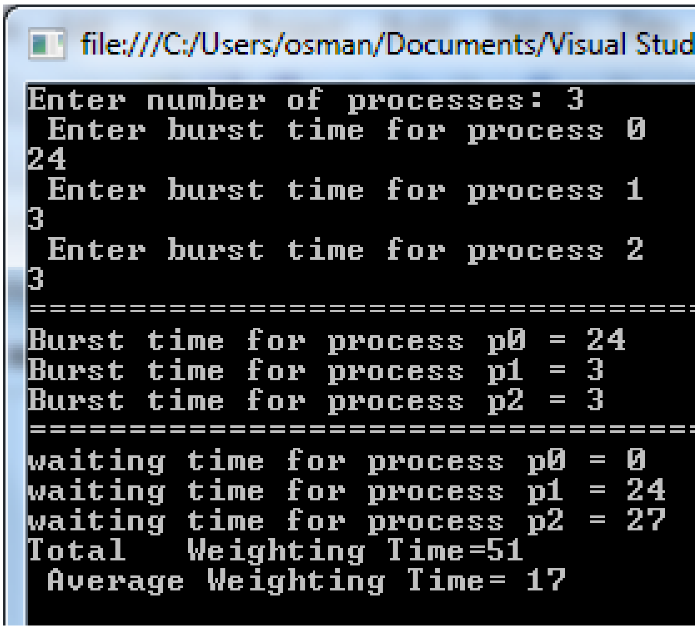
حيث سيكون متوسط زمن إنتظار العملية الأولى هو $wt(0)=0$ ، وزمن إنتظار العملية الثانية هو $w(1) = z(0) + wt(0)$، أي زمن إنتظار العملية الأولى مضافا إليه زمن العملية الأولى. نستخدم التكرار لحساب كل العمليات، فيما يلي الشفرة الكاملة التي تستخدم لإدخال أي عدد من العمليات وحساب زمن إنتظار العمليات (wt) ومتوسط زمن الإنتظار لكل العمليات (awt):

```
Module Module1
Sub Main()
Dim x, y As Integer
Console.Write("Enter number of processes: ")
x = Console.ReadLine()
Dim z(x - 1), z2(x - 1), wt(x - 1) As Integer
For y = 0 To x - 1
   Console.WriteLine(" Enter burst time for process " & y)
   z(y) = Console.ReadLine()
Next
Dim twt As Double = 0.0
For i = 0 To x - 1
   z2(i) = z(i)
   Console.WriteLine("Burst time for p" & i & " = " & z2(i))
Next
wt(0) = 0
For i = 1 To x - 1
      wt(i) = z(i - 1) + wt(i - 1)
Next
For i = 0 To x - 1
Console.WriteLine("waiting time for p" & i & " = " & wt(i))
twt = twt + wt(i)
```

```
Next

Dim Awt As Double = Twt / x
Console.WriteLine("Total   Weighting Time=" & twt)
Console.WriteLine("Average Weighting Time= " & Awt)
Console.Read()
End Module
```

و هنا لقطة من البرنامج و هو يعمل:



يمكن بنفس الطريقة كتابة شفرات لحساب بقية الخوارزميات.

4.7. تمارين

1. أذكر ثلاث أمثلة من خوارزميات الجدولة ؟

2. إذا كان لدينا العمليات التالية مبين فيها ما تحتاجه من وقت داخل المعالج:

العملية	زمن الوصول	زمن العملية
P1	0	9
P2	2	6
P3	4	2
P4	5	4

إذا كان زمن الانتظار =(0+13+5+6)/4 = 6 ، أجب على الآتي:

- ما هي الخوارزمية المستخدمة
- أرسم شكل (Gantt chart) يوضح تنفيذ العمليات داخل المعالج.
- أثبت أن متوسط زمن الانتظار = 6.

3. ماهو الحرمان (starvation) ، وكيف نعالجه ؟

4. بافتراض أن هنالك مجموعة من العمليات موضحة في الجدول التالي:

العملية	زمن الوصول	زمن العملية	الأولوية
P1	0	9	3
P2	2	7	1

P3	4	5	3
P4	6	10	4
P5	7	2	2

المطلوب:

- رسم مخطط جانت (Gantt chart)
- حساب متوسط زمن الإنتظار
- حساب متوسط زمن الإكتمال للعمليات أعلاه باستخدام الخوارزميات التالية:
  - القادم أولا يخدم أولا (FCFS) بدون زمن وصول
  - القادم أولا يخدم أولا (FCFS) مع زمن الوصول
  - الأقصر أولا (SJF) الغير قابلة للتوقف (بدون زمن وصول)
  - الأقصر أولا (SJF) الغير قابلة للتوقف (مع زمن الوصول)
  - الأقصر أولا (SJF) القابلة للتوقف (مع زمن الوصول)
  - الأفضلية (الأهمية) الغير قابلة للتوقف
  - الأولوية (الأهمية) القابلة للتوقف مع زمن الوصول
  - التقسيم الزمني (RR) مع Q=2 بدون زمن وصول
  - التقسيم الزمني (RR) مع Q=2 مع زمن الوصول
  - ما هي أفضل خوارزمية من واقع النتائج التي تحصلت عليها ؟ ولماذا ؟

5. أكتب برنامج بلغة C يقوم بقبول أزمان العمليات الموجودة بالسؤال رقم (4) ثم حساب متوسط زمن الانتظار باستخدام أحدى الخوارزميات التالية: FCFS, SJF, RR, Priority.

6. بافتراض أن هنالك مجموعة من العمليات موضحة في الجدول التالي:

العملية	زمن الوصول	زمن العملية	الأولوية
P1	0	10	3
P2	2	1	1
P3	4	2	3
P4	6	1	4
P5	8	5	2

أحسب متوسط زمن الانتظار باستخدام خوارزميات الجدولة التالية:

1. الأقصر أولا (SJF) الغير قابلة للتوقف (بدون اعتبار لزمن الوصول).
2. الأقصر أولا (SJF) الغير قابلة للتوقف (مع زمن الوصول) .
3. الأقصر أولا (SJF) القابلة للتوقف (مع زمن الوصول).
4. التقسيم الزمني (RR) مع Q=1 (من غير زمن وصول).
5. الأولوية (الأهمية) وذلك باستخدام عمود الأفضلية في الجدول أعلاه.

7. بافتراض أن هنالك مجموعة من العمليات وصلت جميعها في الزمن صفر، وموضح الزمن الذي تحتاجه للتنفيذ في الجدول التالي:

العملية	زمن العملية
P1	7

1	P2
3	P3
2	P4
4	P5

- أرسم مخطط جانت (Gantt chart) يوضح تنفيذ العمليات أعلاه باستخدام خوارزمية التقسيم (RR) في الحالات التالية:
  - عندما تكون Q=1
  - عندما تكون Q=2
  - عندما تكون Q=3
- ما هو زمن إنتظار كل عملية من العمليات أعلاه في كل حالة.
- ما هو الزمن الكلي لكل عملية (turn around time) في كل حالة.
- ما هي الحالة التي تعطي أقل زمن إنتظار ؟ وماذا تستنتج من إجابتك ؟

تنبيه: الزمن الكلي للعملية هو من زمن وصولها (صفر) إلى إكتمالها.

8. أحسب متوسط زمن الانتظار إذا كانت الحصة الزمنية لكل عملية هي 20ملي ثانية.

العملية	الزمن	زمن الوصول
P1	53	0
P2	17	2
P3	68	4
P4	24	6

الباب الخامس:

خيوط التنفيذ

الباب الخامس

خيوط التنفيذ

Thread of Execution

5.1. مدخل

دعنا نضرب مثل تشبيهي يوضح فكرة الخيوط قبل الدخول في تعريفها ومفهومها واستخدامها.

لننظر إلى المسألة الحسابية البسيطة التالية:

5+6+7+8

فإذا طلبنا من أي تلميذ حل هذه المسألة، فإنه سيحلها غالبا في ثلاث خطوات هي:

- حساب 5+6 والناتج هو 13.
- جمع العدد 7 للناتج 13 فيكون الناتج هو 20.
- جمع العدد 8 لناتج العملية السابقة 20 فتكون القيمة النهائية هي 28.

5.1.1. العملية ذات الخيط الواحد (التوالي)

نلاحظ أن الخطوات هذه تتم وراء بعض (بالتوالي). فإذا أفترضنا أن كل خطوة تحتاج ثانية من هذا التلميذ فإنه سينجز العمل في 3 ثوان (أي 13=6+5، 20=7+13، 28=8+20). هنا تتم الخطوات بالتوالي، ولا يستطيع التلميذ القيام بخطوتين في وقت واحد، وهذا يشبه العملية العادية (ذات الخيط الواحد).

5.1.2. العملية متعددة الخيوط (التوازي)

الآن لو احضر هذا التلميذ أثنين من زملائيه ليساعدوه في حل هذه المسألة، فيمكن للتلميذ الأول أن يحسب 6+5، والتلميذ الثاني يحسب 8+7 ، في وقت واحد (الخطوتين في ثانية واحدة)، ويقوم التلميذ الثالث بجمع النتيجتين ليحصل على القيمة النهائية في

ثانية، هنا نجد أن المسألة حلت في ثانتين بدلا من ثلاث ثوان (أي 33% أسرع من الأولى). نلاحظ أن الخطوتين الأوائل قد تمتا في وقت واحد، لكن الخطوة الثالثة فلا يمكن أن تتم إلا بعد إنتهاء الخطوتين السابقتين لأنها تعتمد علي نتائجهما. من هنا نسنتج:

- أن القيام بأكثر من عمل في وقت واحد يعني عملية لها أكثر من خيط.
- لا يمكن للخيوط أن تعمل بالتوزاي إذا كانت تعتمد على بعضها، لابد من أن تكون الخيوط مستقلة عن بعضها لتعمل معا في وقت واحد، وإلا فلن نستفيد من وجود أكثر من خيط في العملية.

5.2. تعريف الخيط

العملية في وضعها الطبيعي تنفذ عمل واحد، ونطلق عليها عملية أحادية الخيط (single thread). ولكن ماذا لو أردنا من العملية أن تنفذ أكثر من عمل في نفس الوقت، هنا لابد من أن نجعل العملية متعددة الخيوط (multiple threads)، الشكل (5-1). حيث تشتغل كل الخيوط التي توجد في العملية في وقت واحد بالتوازي مما يحقق التزامنية (concurrency). تتشارك خيوط العملية الواحدة في بنية معلومات واحدة (PCB)، لكن لكل خيط بنية معلوماته الخاصة به التي تختلف عن بقية بينية معلومات الخيوط الأخرى التي معه في نفس العملية. بنية معلومات الخيط تحتوي على :

- المكدس (stack).
- المسجلات (register).
- عداد البرامج.
- حالة الخيط.

أحيانا نطلق على الخيط عملية خفيفة (lightweight process). ذلك لأن الخيط يمتلك الكثير من خصائص العمليات.

لا ترتبط الخيوط بأي مورد لذلك فإن إنشاءها وتدميرها أسهل من إنشاء وتدمير العمليات. وقد يكون إنشاء الخيط في بعض الأنظمة أسرع بمئة مرة من إنشاء العملية.

5.2.1. مثال تشبيهي

إذا تعاملت مع شركة معمارية وطلبت منها تنفيذ عمل معين، فأنت في تعاملك مع الشركة لا تهتم ولا تعرف كم عامل سينفذ عملك، فالشركة تشبه العملية، والعمال هم (الخيوط) داخل الشركة (العملية). إذا نفذ العمل عامل واحد فأنت تتعامل مع الشركة كعملية تقليدية ذات خيط واحد (النظام القديم)، فعلى العامل هنا أن ينفذ العمل بالتوالي جزئية تلو الأخرى، أما إذا نفذ العمل عدد من العمال فأنت تتعامل مع شركة (عملية) ذات خيوط متعددة، حيث ينفذ العمال العمل بالتوازي، كل عامل يقوم بجزئية من العمل في وقت واحد. والفرق واضح بين الاثنين.

تستخدم معلومات الشركة حينما تتعامل معها (يشبه PCB)، وداخل الشركة توجد معلومات عن كل عامل (معلومات الخيط)، حيث يتشارك كل العمال في عنوان الشركة، ولكن يختلفوا في عناوينهم الخاصة والتي تميزهم عن بعضهم البعض.

5.3. أنواع الخيوط

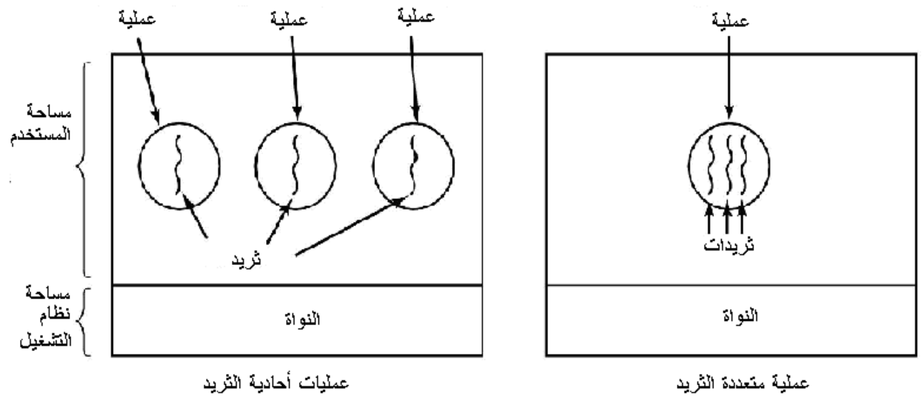
تختلف الخيوط في طريقة الإدارة والدعم، فهنالك خيوط تدار وتدعم بواسطة نظام التشغيل وهنالك ما يدار بواسطة المستخدم، وكذلك توجد خيوط يمكن أن تدار بواسطة لغة برمجة معينة، أدناه سنفصل الفرق بين كل نوع مع إعطاء مثال من كل نوع.

5.3.1. خيط المستخدم (**user thread**)

قد يتعامل نظام التشغيل (خارجيا) مع العملية كعملية واحدة دون معرفة الخيوط الداخلية التي توجد فيها، بينما داخليا تنفذ العملية مجموعة من الخيوط في وقت واحد. هذا النوع من الخيوط يسمى خيط المستخدم، أي أن إنشاء الخيط والتعامل معه يتم في مستوى المستخدم (user level) ولا شأن لنظام التشغيل به، فهو يتعامل مع العمليات دون التمييز بين التي بها خيط واحد والتي بها العديد من الخيوط.

يتم تطبيق خيط المستخدم بواسطة مكتبات. من المكتبات التي تدعم خيط المستخدم Pthreads، حيث توفر هذه المكتبة كل ما يتعلق بالتعامل مع الخيط من

إنشاء، وإنهاء، جدولة، وإدارة، دون أي مساعدة أو تدخل من نظام التشغيل، ودون الإرتباط بلغة برمجة معينة. عيب هذا النوع أن نظام التشغيل لا يتعامل مع الخيوط الموجودة بالعملية، لذلك عندما يحجز عملية معينة، فإنه يحجز العملية ككل بما فيها من خيوط ولا يستثنى أي خيط داخل العملية من الحجز، رغم أن بعض الخيوط قد تكون قادرة على العمل. أما ميزته فهي سرعة وسهولة إنشاء وإدارة الخيوط.



شكل رقم (5-1): على اليسار: ثلاث عمليات كل واحدة تحتوي على خيط واحد.

على اليمين: عملية واحدة تحتوي على ثلاث خيوط.

5.3.2. خيط النواة

هنا يتم دعم الخيط مباشرة بواسطة نظام التشغيل فهو الذي يوفر طرق التعامل مع الخيط من إنشاء وإنهاء، جدولة وإدارة. ولأن إدارة الخيط تتم بواسطة نظام التشغيل فهذا يجعل خيط النواة بطئ نوعا ما مقارنة بخيط المستخدم. ولكن إذا تم توقيف خيط لسبب ما، فيمكن لخيوط أخرى في نفس العملية أن تعمل دون أن تتأثر بحجز رفيقاتها. من أمثلة نظم التشغيل التي تدعم خيوط النواة، نظام التشغيل ويندوز، الذي يمتلك مكتبة تدعم الخيوط هي مكتبة win32. حيث يمكنك استخدام windows.h، ثم إنشاء الخيط بالأمر CreateThread().

هنالك نظم تشغيل تدعم النوعين، خيط المستخدم وخيط النواة.

5.4. خيوط جافا

لغة جافا تعتبر من اللغات التي توفر دعم الخيط على مستوى اللغة، فهي توفر مكتبة كاملة لإنشاء وإدارة الخيوط.

5.4.1. إنشاء الخيط

أبسط طريقة لإنشاء الخيط في الجافا يكون بوراثة فئة الخيط المسماة (Thread) والموجودة بمكتبة جافا. مثلا الأمر التالي:

```
class th extends Thread
```

يمكنك من وراثة الصف Thread، حيث يكون لديك الصف th الذي يتعتبر صف خيط. يحتوي الصف th على دالة تسمى run، هذه الدالة هي التي تنفذ الخيط، وقد ورثناها من الصف Thread. سنضع ما نريد تنفيذه داخل هذه الدالة، مثلا:

```
public void run(){
    System.out.print("Inside thread"):}
     }
```

بعدها ننشئ كائن من الصف th، مثلا:

```
th t1 = new th( )
```

الآن أصبح لدينا خيط هو t1، يمكن تشغيله بالأمر:

```
t1.start( )
```

تنبيه: الخيط t1 يعتبر الخيط الثاني في العملية ذلك لأن العملية في الأساس لديها خيط واحد يعتبر الخيط الرئيسي. فكل عملية تحتوي عادة على خيط رئيسي واحد، إذن الآن لدينا خيطين في العملية. تشغيل خيط العملية الأساسي يتم في main. البرنامج (1-5) هو برنامج صغير يوضح كيفية إنشاء ثريد وتشغيله من داخل main.

```
class th extends Thread{//
```

```
public void run(){
      System.out.print("Inside thread"):}
}
public static void main(String args[ ]){
       th t1 = new th( ); // إنشاء خيط
       t1.start( ); // تشغيل الخيط
      System.out.print("Inside main");}
}
```

برنامج رقم (5-1)

5.4.2. التعامل مع الخيط

توفر جافا طرق (دوال) للتعامل مع الخيط، مثل:

Suspend()	توقيف الخيط، ويمكن تشغيله مرة ثانية بالأمر Resume()
Sleep()	توقيف (تنويم) الخيط لفترة زمنية محددة يعود ليعمل مرة أخرى بعد انقضائها.
Resume()	تشغيل الخيط مرة ثانية بعد توقيفه بالأمر suspend().
Stop()	إنهاء الخيط (قتله)، حيث يصبح الخيط ميت ولا يمكن تشغيله مرة ثانية.

عدل البرنامج (5-1)، لتطبق عليه الدوال أعلاه.

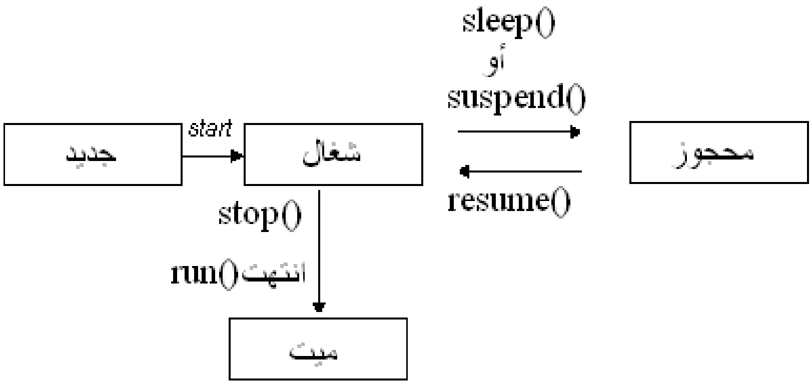
5.5. حالات الخيط

للخيط حالات ينتقل بينها أثناء التنفيذ، الشكل (5-2)، هي:

- جديد (new): عند إنشائه بالأمر new. في البرنامج (5-1)، يعتبر الخيط th في حالة جديد عند استخدام الأمر:

```
th t1 = new th();
```

- شغال (منفذ) Runnable: الأمر start() يحجز مساحة بالذاكرة للخيط، ويستدعى الطريقة run() بالكائن t1 (في المثال (5-2))، هنا نقول أن حالة الخيط t1 هي Runnable.
- محجوز (blocked) أو غير شغال (not Runnable): إذا كان الخيط يقوم بعملية دخل أو خرج، أو تم استدعاء suspend() أو sleep().
- ميت (Dead): يصبح الخيط ميتاً إذا انتهى عمل الطريقة run() أو استدعينا stop().



شكل رقم (5-2): حالات الخيط.

من الصعب معرفة حالة الخيط، ولكن قد تعرف هل الخيط حي يرزق أم لا باستخدام الطريقة isAlive() والتي ترجع قيمة منطقية (نعم (حي) ، أم لا (مات)).

5.6. التحول بين العمليات ***Context Switch***

لكل عملية بنية بيانات تسمى (PCB)، بغض النظر أن العملية هل فيها خيط واحد أم أكثر. وكل الخيوط الموجودة في العملية الواحدة تشترك في بنية العملية (PCB)، بينما يكون لكل خيط بنيته الخاصة (الشكل (5-3)) التي يستخدمها عندما يتحول من حالة التنفيذ إلى حالة أخرى. بنية معلومات الخيط تتكون من:

- المسجلات بما فيها مسجل عداد البرامج (PC).
- المكدسة (stack).

نلاحظ هنا أن بنية الخيط صغيرة الحجم مقارنة مع بنية العملية وبالتالي تحول المعالج بين الخيوط يكون أسرع وأبسط من تحول المعالج بين العمليات.

انتقال المعالج بين العمليات أو الخيوط تتم بإخراج عملية من المعالج (توقيفها أو حجزها)، وإدخال عملية أخرى للمعالج للتنفيذ، هنا نقول أن المعالج انتقل من تنفيذ العملية الأولى إلى تنفيذ العملية الثانية. هذا الانتقال أو التحول يسمى (context switching) ويعتبر مكلف زمنيا، فلكي يتم الانتقال لابد من حفظ معلومات العملية (PCB) المنفذة حاليا (المنتقل منها)، وتحميل معلومات العملية (PCB) التي ستدخل المعالج (المنتقل إليها).

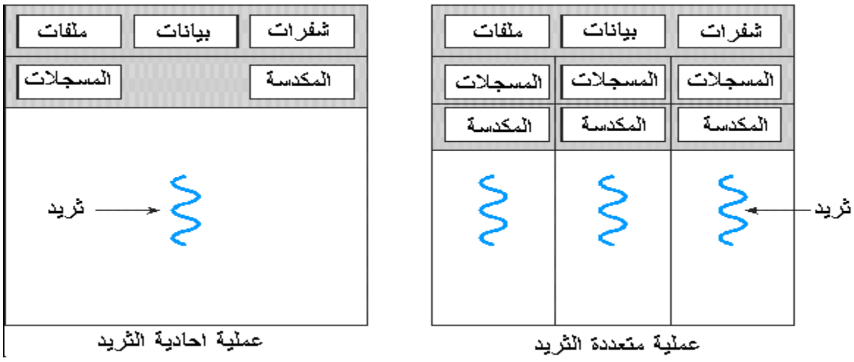
ويحدث نفس الأمر في الانتقال بين الخيوط، لكن الفرق هنا أن معلومات الخيط التي ستحفظ ومعلومات الخيط الآخر التي ستحمل قليلة وبالتالي فإن الزمن المستغرق لحفظ وتحميل بيانات الخيط أقل بكثير من الزمن المستغرق لحفظ وتحميل بيانات العملية، لذلك يعتبر زمن التحويل بين الخيوط أقل من زمن التحويل بين العمليات.

عملية حفظ بيانات العملية الموقفة وتحميل بيانات العملية المراد تشغيلها يستغرق وقتا من نظام التشغيل ويسمى زمن التحويل (context switching time). ويعتمد الزمن المستغرق في التحول من عملية إلى أخرى على العتاد المستخدم.

ملحوظة

في الأنظمة أحادية المعالج (حاسب بمعالج واحد)، فإن المعالج لا يستطيع تشغيل أكثر من عملية في اللحظة الواحدة، ولكن يمكن الاستفادة القصوى من المعالج

بتحميل عدد من العمليات في الذاكرة ثم جعل المعالج ينتقل بين هذه العمليات بسرعة عالية، وكلما احتاجت عملية انتظار حدث يقوم نظام التشغيل (مدير المعالجة) بإخراجها ثم تشغيل عملية أخرى مكانها وهكذا يظل المعالج مشغولا. بينما يبدو للمستخدم أن المعالج يشغل عدد من العمليات معا (تعدد المهام).



شكل رقم (5-3): بنية العملية تكون مشتركة بين الخيوط، ولكل خيط بنيته الخاصة.

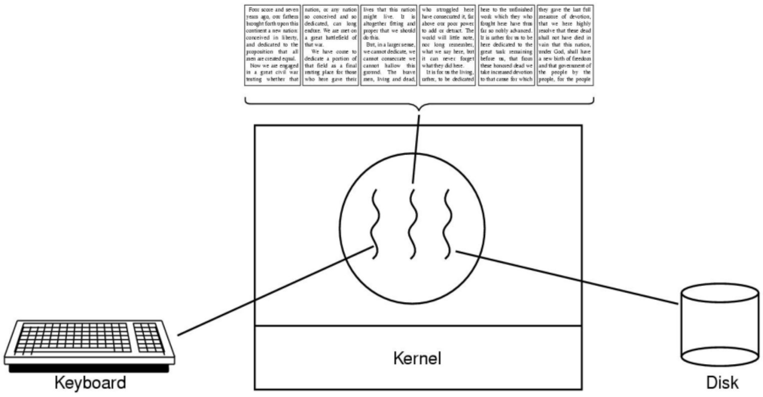
5.7. استخدامات الخيط

كل البرامج الحديثة التي نتعامل معها مكونة من مجموعة من الخيوط، فمحرر النصوص ، والمتصفح، الرسام ، وكل البرامج التي حولك تجدها مكونة من عدد من الخيوط. ذلك لأن معظم التطبيقات بها العديد من الأشياء التي تحدث في وقت واحد.

5.7.1. محرر النصوص

الشكل رقم (5-4) يوضح مثال لعملية (برنامج محرر نصوص) يحتوي على ثلاث خيوط، وكل خيط ينفذ مهمة معينة، فهنالك خيط يستجيب للمدخلات عبر لوحة المفاتيح والماوس وينفذ الأوامر التي ترد عبرها مثل الانتقال لصفحة معينة. بينما هنالك خيط ثاني يقوم بعملية إعادة شكل الوثيقة ، فإذا قمت بحذف جمله سيقوم هذا الخيط بإعادة ترتيب النص كاملا بعد حذف الجملة، وهنالك خيط ثالث يقوم بعملية الحفظ التلقائي (كل فترة).

مثلا لو كان برنامج محرر النصوص يعمل بخيط واحد ، فلن تستطيع الانتقال إلى صفحة مع ظهور شكل البيانات المعدلة في نفس الوقت، وعندما يعمل الحفظ التلقائي لابد من توقف كل شيء حتى يكتمل الحفظ.



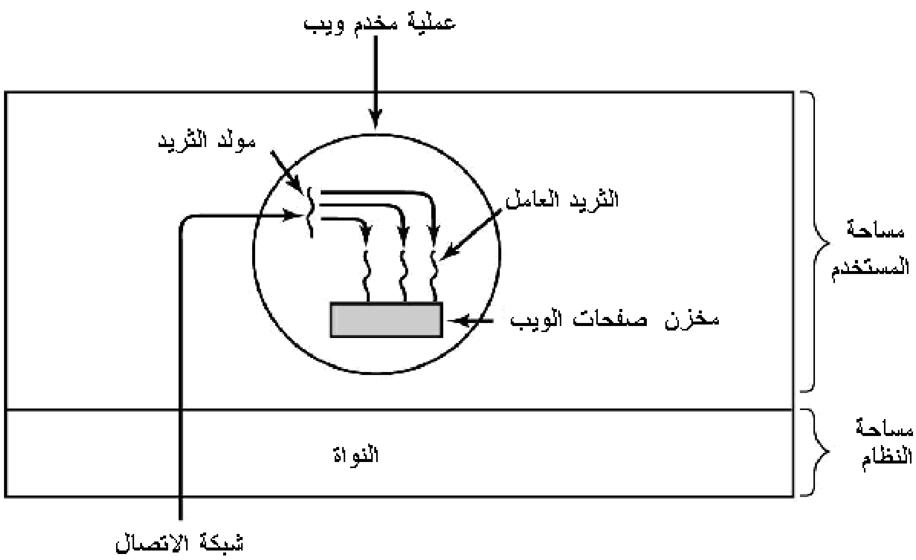
شكل رقم (5-4): برنامج محرر نصوص مكون من ثلاث خيوط.

5.7.2. مخدم الويب

وظيفة مخدم الويب (web server) هي الرد على طلبات المتصفحين وإرسال ما يريدون من صفحات إلى أجهزتهم. ويوجد في كل جهاز مخدم ويب على الإنترنت مثلا وفيه عدد من المواقع، قد يتصل أكثر من شخص بهذا الجهاز ويطلب عرض أكثر من صفحة في نفس الوقت، هنا ينشي مخدم الويب خيط لكل متصل يتحاور معه ويرسل له طلباته.

ماذا لو كان مخدم الويب يعمل بخيط واحد ؟

إذا اتصل مائة عميل بهذا المخدم في وقت واحد، سيكون هنالك خيط واحد لخدمة هؤلاء العملاء، واحد تلو الآخر (بالتتالي)، فإذا أفترضنا أن كل طلب يستغرق دقيقة واحدة، فإن العميل رقم مائة ستتم خدمته بعد ساعة وأربعون دقيقة. هل ستستخدم الإنترنت إذا كان المخدم يعمل بهذه الطريقة ؟



شكل رقم (5-5): عملية مخدم ويب تنشي خيط لكل طلب.

5.8. استخدام ***Pthreads***

هنا سنوضح مثال بسيط لاستخدام مكتبة Pthreads ليتضح الفرق بين استخدام الخيط في جافا (على مستوى لغة البرمجة) وبين استخدامه في مكتبة غير مرتبطة بلغة برمجية معينة. Pthreads هي مكتبة متوفرة في لينكس قمنا باستخدامها مع لغة C لتوضيح كيف ننشئ خيط وكيف نديره بواسطة هذه المكتبة.

5.8.1. إنشاء خيط

لإنشاء خيط نحتاج استخدام الدالة pthread_create() والتي تحتوي على المعطيات التالية:

- المعطى الأول نحصل من خلاله على تعريف الخيط (thread identifier).
- المعطى الثاني مؤشر إلى مكان الكائن الذي يحدد صفات الخيط، إذا استخدمت null هنا فهذا يعني أنك تريد الصفات الافتراضية للخيط.
- المعطى الثالث هو مؤشر إلى مكان الدالة التي ينفذها الخيط.

- المعطى الأخير هو القيم (argument) التي نريد تمريرها إلى الدالة المنفذة داخل الخيط، إذا لم يكن لديك قيم تريد تمريرها إلى الدالة يمكنك كتابة null في هذه الخانة.

مثلا لو كتبت شفرة الدالة بهذه الطريقة:

```
pthread_create(&th_ID, NULL, th_fun, &value);
```

فهذا يعني أنني أريد إنشاء خيط يخزن تعريف هذا الخيط في المتغير th_ID ، ويستخدم هذا الخيط الصفات الافتراضية، وينفذ هذا الخيط الدالة th_fun التي تكون مدخلاتها value.

إذا أردت إنشاء ثلاث خيوط فعليك استدعاء الدالة ()pthread_create ثلاث مرات.

أيضا نستدعي الدالة ()pthread_join مع كل خيط قمنا بإنشائه لضمان إنتهاء الخيوط قبل إنتهاء الدالة الرئيسية main.

5.8.2. مثال **pthread1.c**

البرنامج رقم (5-2) يقوم بإنشاء خيطين، حيث يقوم كل خيط بطباعة رسالة على الشاشة.

نترجم البرنامج كالآتي:

C compiler: cc -lpthread pthread1.c

أو

C++ compiler: g++ -lpthread pthread1.c

ثم ننفذ البرنامج كما يلي:

```
./a.out
```

مخرج البرنامج:

```
Thread 1
Thread 2
Thread 1 returns: 0
Thread 2 returns: 0
```

برنامج (2-5)

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
void *p_message( void *ptr );


main(){
    pthread_t thread1, thread2;
    char *m1 = "Thread 1";
    char *m2 = "Thread 2";
    int  r1, r2;


/* ننشئ خيطين، كل خيط ينفذ نفس الدالة لكن بمعطى مختلف */
/* إنشاء الخيط الأول */
r1 = pthread_create( &thread1, NULL, p_message, (void*) m1);

/* إنشاء الخيط الثاني */
r2 = pthread_create( &thread2, NULL, p_message, (void*) m2);
/* نجعل الدالة الرئيسية تنتظر حتى يكتمل تنفيذ الخيطين */
    pthread_join( thread1, NULL);
    pthread_join( thread2, NULL);
    printf("Thread 1 returns: %d\n",r1);
    printf("Thread 2 returns: %d\n",r2);
    exit(0);

}
```

```
/* الدالة التي تنفذ داخل كل خيط مع مدخل مختلف */
void *print_message_function( void *ptr )
{     char *message;
    message = (char *) ptr;
    printf("%s \n", message);

}
```

5.8.3 إنهاء خيط

لإنهاء خيط نستخدم الدالة pthread_cancel()، لكن عليك التأكد من أن الخيط الذي تريد إنهاؤه لا يستخدم موارد قبل إنهاؤه. مثلا إذا كان الخيط يحجز مساحة بالذاكرة وقمنا باستدعاء دالة الإنهاء pthread_cancel() سنفقد مكان هذه الذاكرة وستظل محجوزة بلا فائدة (memory leak).

5.9. تمارين برمجية

أكتب البرامج التالية باستخدام مكتبة pthread أو بالجافا:

برنامج (1)

يقوم بطباعة سلسلة فيبوناكي (Fibonacci series) أي:

1,2,3,5,8,13,…

يجب أن يعمل البرنامج كالآتي:

ينفذ المستخدم البرنامج ثم يدخل رقم يمثل عدد الأعداد التي نريد من البرنامج أن يولدها من سلسلة فيبوناكي، هنا ينشئ البرنامج خيط يقوم بعملية توليد الأعداد.

Enter number of Fibonacci numbers to generate ? 5

1,2,3,5,8

برنامج (2)

يقوم بطباعة الأعداد الأولية من 1 إلى رقم معين. حيث يطلب البرنامج من إدخال الرقم الذي نريد توليد أعداد أولية أقل أو تساويه، ثم ينشئ البرنامج خيط يقوم بعملية توليد الأعداد الأولية.

Enter number to generate primes less than or equal to ? 5

2,3,5

5.10. تمرين غير محلولة

1. عرف الخيط ؟
2. ما الفرق بين الخيط والعملية ؟
3. لا ترتبط الخيوط بأي مورد، وضح ذلك ؟
4. إنشاءه الخيط وتدميرها أسهل من إنشاء العمليات لمذا ؟
5. "إنشاء الخيط أسرع بمئة مرة من إنشاء العملية" ، أثبت أن هذه العبارة صحيحة (أبحث في الإنترنت عن ذلك) ؟
6. ما الفرق بين خيط المستخدم وخيط النواة ؟
7. هل يعتبر الخيط في جافا خيط مستخدم ؟
8. هل تعتبر pthreads خيط مستخدم ام نواة ؟
9. أذكر مثال لنظام تشغيل يدعم خيط المستخدم ومثال لنظام تشغيل يدعم خيط النواة ؟
10. ما هي مكونات بنية معلومات الخيط التي يحتاج لحفظها عند التحول من حالة التنفيذ إلى حالة أخرى؟
11. أذكر حالات الخيط المختلفة ؟
12. أذكر دوال الخيط في جافا والتي تستخدم في تشغيل وتوقيف وإنهاء الخيط ؟

الباب السادس:

التزامن

الباب السادس

التزامن (synchronization)

6.1. مقدمة

إذا كنت تبرمج على حاسب شخصي وتكتب برامج عادية، فسيكون همك الأكبر هل المخرج أو الناتج الذي يظهره البرنامج وهل هو المطلوب أم لا ؟

ولكن للتطور تبعاته، فإذا صممت برنامج ليعمل في شبكة ويتشارك في البيانات مع برامج أخرى، فستظهر هموم أخرى غير هموم النتائج الصحيحة، مثل هل البيانات متوافقة؟ وهل ما قراءته من معلومات هو آخر ما تم تحديثه، وهل على برنامجك الانتظار في لحظة معينة حتى يتزامن عمله مع بقية البرامج التي تتشارك معه في البيانات أو البرامج التي تتعاون معه في انجاز مهمة مشتركة؟ و ما إلى ذلك من مشاكل جلبها علينا التزامن والعمل المشترك عبر الشبكات.

6.2. مفهوم التوازي *Concurrency*

نقصد بالتوازي تنفيذ أكثر من عملية في وقت واحد. قد تكون هذه العمليات المتوازية مستقلة عن بعضها البعض (independent processes) أو متعاونة مع بعضها البعض (cooperative processes). مثلا قد أقوم بتشغيل محرر النصوص (WinWord) لأكتب عليه، واستمع إلى مادة صوتية بواسطة مشغل الوسائط (media player)، ويكون المتصفح ينزل ملفات من الانترنت، هذه العمليات تنفذ بالتوازي في وقت واحد ولكنها مستقلة عن بعضها البعض.

التوازي قد يطبق على مستوى العمليات (concurrent processes) أو على مستوى الخيوط (concurrent threads).

العملية المستقلة هي التي لا تؤثر ولا تتأثر بالعمليات التي تعمل معها في نفس الوقت. العملية تكون متعاونة إذا أثرت و/أو تأثرت بالعمليات الأخرى التي تعمل معها بالتوازي.

6.3. تعاون العمليات

إذا احتاجت العمليات المتوازية أن تتشارك في بعض البيانات أو أن تؤدي عملا مشتركا، فلابد لها من أن تتعاون مع بعضها البعض. هذا التعاون يتطلب اتصال بين العمليات (أو الخيوط) وعملية تزامن (synchronization).

الاتصال بين العمليات يتم عن طريق متغيرات مشتركة (shared variables) أو بتبادل الرسائل (message passing). والتزامن نحتاجه ليتم الاتصال في الوقت المناسب، ويتحقق بوضع نقاط تزامن للعمليات بحيث إذا سبقت عملية بقية العمليات ووصلت نقطة تزامن، عليها أن تنتظر بقية العمليات الأخرى لتصل هذه النقطة، وإلا سيكون لدينا نتائج غير متوقعة.

لماذا تحتاج العمليات المتوازية التعاون فيما بينها ؟ لعدة أسباب، مثل:

- طلب عملية لخدمة من عملية أخرى: في هذه الحالة على العملية التي طلبت الخدمة انتظار العملية التيُطلب منها الخدمة حتى تفرغ من أدائها. مثلا إذا طلبت عملية من نظام التشغيل معلومة معينة من القرص الصلب، ستظل هذه العملية في انتظار المعلومة حتى تصلها من القرص إلى الذاكرة، أو إذا طلب المتصفح من مخدم الويب بالانترنت موقع معين فعلى المتصفح الانتظار حتى يرسل المخدم الموقع المطلوب، فيقوم المتصفح وقتها بعرضه على المستخدم.

- زيادة سرعة التنفيذ: وذلك بتقسيم مهمة واحدة كبيرة على عدة عمليات يتم تنفيذها بالتوازي. قد تسبق عملية في تنفيذها عمليات أخرى معها، فإذا تركنا هذه العملية لتكمل تنفيذها دون انتظار العمليات الأخرى فقد يتسبب هذا في نتائج غير متوقعة أو غير صحيحة، لذلك لابد من وضع نقاط تزامن لا تتعداها العمليات السريعة ما لم تصل بقية العمليات هذه النقاط. مثلا إذا استخدمنا الأنابيب الانسيابية (pipeline) بين عمليات فهذا يعني أن مخرج العملية الأولى سيكون مدخل للعملية الثانية، في هذه الحالة لا يمكن للعملية الثانية أن تسبق العملية الأولى. مثلا الأمر التالي (في ينكس):

```
$ ls | wc
```

هو أمر لتنفيذ عمليتين هما ls ، التي تعرض محتويات الدليل الحالي، و wc لعد الكلمات، هاتان العمليتان ستنفذان في وقت واحد (بالتوازي)، وسيتم بينهم تزامن (ستنتظر العملية الثانية مخرجات العملية الأولى لتعمل عليها (تحسب عدد كلمات المخرج)).

- قد يكون من الملائم للعمليات أن تعمل مع بعضها لانجاز العمل، مثلا في نظام الوسائط المتعددة يمكن أن تقوم عملية بإحضار أجزاء المادة الوسائطية من القرص ووضعها في ذاكرة مؤقتة (خازن (buffer))، بينما عملية أخرى تأخذ المادة من الخازن وترسلها إلى جهاز الذي يصدر الصوت/الصورة. هنا تعتبر العملية الأول منتج (producer) بينما العملية الثانية مستهلك (consumer). وعلى المنتج العمل على جعل الخازن ممتلئا دوما بحيث لا يتوقف المستهلك عن العمل (لا يجد الخازن فارغا)، لأن المستهلك إذا وجد الخازن فارغا سيتوقف عن العمل وينتظر وصول بيانات للخازن مما يتسبب في تقطع مقطع الفيديو/الصوت أو عدم وضوحه.

أيا كان نوع التعاون بين العمليات، فيجب أن يكون هنالك:

- تزامن بينها.
- اتصال فيما بينها.

6.2.2. الحجز غير المتوقع للعمليات

قد يقوم المجدول (في نظام التشغيل) بحجز أي عملية في أي وقت ولأي سبب، هذا الحجز سيوقف العملية التي تنفذ حاليا في المعالج، ويدخل عملية أخرى مكانها (بالطبع سيكون هنالك تحول (context switching)، حيث يتم حفظ معلومات العملية المحجوزة ويتم تحميل معلومات العملية التي سيتم تنفيذها).

لا يستطيع المبرمج معرفة متى سيقوم المجدول بحجز برنامجه (عمليته).

هذا يعني أن البرنامج قد يتوقف في أي وقت (عندما يحجز)، ثم فيما بعد قد يواصل من آخر نقطة أوقف فيها (بعد فك حجزه). قد يتسبب هذا الحجز في مشكلة إذا كانت العملية المحجوزة ضمن عمليات متعاونة.

6.4. النزاع (*competition*)

قد تكون للعمليات المتعاونة بيانات مشتركة تستطيع الوصول إليها. إذا لم يكن هنالك تحكم في طريقة الوصول لهذه البيانات ستكون النتيجة وصول غير منظم لها أو تنازع حولها وبالتالي نتائج غير صحيحة. هذا الوصول غير المنظم للبيانات المشتركة يسمى حالة السباق (race condition). حيث تتسابق العمليات في تغيير قيمة البيانات المشتركة.

أيضا قد تكون العمليات مستقلة لكنها تتصل مع بعضها لإستخدام موارد مشتركة، مثلا إفترض أن لدينا عمليتان، كل عملية تريد طباعة ملف على الطابعة (المشتركة)، لابد لهما من الإتصال بينمها ليقررا من يستخدم الطابعة أولا، وعلى العملية الثانية الإنتظار جتى تفرغ الأولى من الطباعة. هذا يسمى هذا تنافس العمليات (process competition).

6.4.2. مشاكل النزاع

فيما يلي نسرد بعض الأمثلة التي تحدث فيها مشاكل نتيجة للإيقاف غير المتوقع لبعض العمليات بواسطة المجدول.

6.4.2.1. مشكلة تعديل ملف على الخادم الملفات (file server)

إفترض أن لدينا شبكة حواسيب حيث يتم تخزين كل ملفات المستخدمين في خادم الملفات. إذا كان هنالك مستخدمين يريدان تعديل ملف مشترك بينهما في نفس الوقت، سيقوم كل مستخدم بفتح الملف في جهازه (نسخة من الملف)، ثم يقوم كل منهما بتعديل نسخته، هذا التعديل لن يتم على الملف الأصيل بالخادم وإنما في نسخة كل مستخدم. وعندما يقوم المستخدم بالحفظ سيغير ذلك في الملف الأصلي. المشكلة هي أنه عندما يقوم المستخدم الثاني بحفظ نسخة ملفه في الخادم سيكتب فوق تعديلات المستخدم الذي حفظ ملفه أولا (overwrite). هذه المشكلة ينتج عنها مخرجات خاطئة.

الصحيح هو أنه عندما تعمل عمليتنان في وقت واحد يجب أن تكون النتيجة متشابهة بغض النظر أي عملية أنتهت أول. ولكن عندما يؤثر ترتيب تنفيذ العمليات في النتيجة يسمى هذا حالة السباق (race condition). لأن ذلك يمثل سباق بين العمليات لنعرف من ينتهي أولا.

6.4.2.2. مشكلة نظام الحجز الموزع

إذا كان هنالك نظام حجز على خطوط طيران معينة، يعمل على عدة فروع. وكان هنالك مقعد واحد فارغ في رحلة ما، وجاء عميل لحجز هذا المقعد في الفرع أ، وفي نفس الوقت كان هنالك عميل بالفرع ب ، يريد حجز نفس المقعد، وقام الموظفين في الفرعين بعملية طلب حجز للمقعد في نفس الوقت، ماذا يحدث ؟

العملية الأولى في الفرع أ قامت بالتأكد من وجود مقعد فارغ، وهذا ما فعلته أيضا العملية التي نفذت في الفرع ب، فالعمليتان قامتا باختبار وجود مقعد فارغ، ثم أجاب النظام للعمليتين "بنعم يوجد مقعد واحد فارغ"، في هذه الإثناء قام المجدول (بنظام التشغيل) بتوقيف إحدى العمليتين لسبب ما، فقامت العملية الأخرى بحجز المقعد، ثم بعد فك توقيف العملية الأولى ستكمل عملها وتنفذ الأمر الذي يلي اختبار وجود مقعد فارغ (فهي اختبرت المقعد ووجدته فارغا قبل توقيفها). فتقوم العملية بحجز المقعد الذي حجز من قبل، وبهذا يتم حجز المقعد مرتين.

6.4.2.3 مشكلة الحساب المشترك

إذا كان هنالك حساب مفتوح ببنك ما، وكان هذا الحساب مشترك بين عميلين، فقد يتفق وأن يطلب العميلين سحب مبلغ من المال من الحساب المشترك في وقت واحد من فرعين مختلفين. هنا ستنفذ عمليتين على الحساب المشترك، حيث ستقوم العمليتين باختبار الرصيد (نفترض أنه كان 1500 دولار)، ثم إذا أوقفت إحدى العمليتين بواسطة المجدول لسبب ما، ونفذت العملية الأخرى سحب المبلغ المطلوب (مثلا 100 دولار)، سيكون الرصيد الآن 1400 دولار. بعد أن يتم فك حجز العملية الثانية ستواصل من بعد آخر أمر كانت قد نفذته، وهو اختبار الرصيد (1500 دولار)، وستنفذ السحب وتخصم المبلغ من الرصيد (مثلا إذا كان المبلغ المراد سحبه هو 200 دولار ، فسيكون الرصيد المتبقي هو (1300 دولار). وتعتبر هذه العملية خطأ، ولو عمل البنك بهذه الطريقة سيغلق أبوابه قريبا.

النتيجة الخاطئة التي وصلنا إليها كان سببها الوصول غير المنظم لقاعدة البيانات وإجراء تعديل على الحساب المشترك بصورة غير منسقة (غير تزامني) مما

نتج عنه خطأ يسمى حالة السباق (race condition). حل مشكلة مثل هذه يتم بالتأكد من أن عملية واحدة فقط في لحظة معينة تقوم بتعديل البيانات المشتركة.

6.4.2.4. حالة السباق (race condition) والمنطقة الحرجة (critical section)

المشاكل التي سرندناها مسبقا معظمها تعاني من حالة السباق، ولنوضح حالة السباق وطريقة حلها سنأخذ المثال التالي:

إذا كان لدينا متغير x، مشترك بين عمليتين، وقامت كل عملية بالآتي:

- قراءة قيمة المتغير x (read x).
- زيادة قيمة المتغير بواحد (x=x+1).
- حفظ قيمة المتغير الجديدة (writet x).

المشكلة ستظهر عندما تقوم أحد العمليات بقراءة قيمة المتغير، ثم تقوم العملية الثانية بقراءة قيمة المتغير قبل أن تقوم العملية الأولى بحفظ القيمة الجديدة في المتغير. فستكون النتيجة النهائية بعد تنفيذ العمليتان أن المتغير سيزيد بواحد بدلا من 2.

الحل هو أن نمنع أي عملية أخرى من الوصول للمتغير x إذا كانت هنالك عملية تستخدم هذا المتغير.

المنطقة التي تتعامل مع المتغير المشترك في العملية نسميها المنطقة الحرجة (critical section). ستكون منطقة المتغير المشتركة هي المنطقة الحرجة، ولحل مشكلة حالة السباق يجب أن نتأكد من أن هنالك عملية واحدة تنفذ داخل المنطقة الحرجة. ولا يسمح للعملية الثانية بالدخول إلى المنطقة الحرجة إلا بعد خروج العملية الأولى منها.

6.5. مشاكل التزامن الكلاسيكية

هنا سنتطرق على بعض المشاكل الناجمة عن التزامن والتي تحدث في العمليات المتوازية وهي مشاكل مشهورة وتستخدم لاختبار أي نظام جديد تزامني مقترح.

6.5.1. مشكلة القراءة والكتابة (reading and writing problem)

في مشكلة الحجز الموزع ومشكلة الحساب المشترك بالبنك، تعتبر عملية التعديل على قاعدة البيانات، عملية كتابة، بينما عملية الحصول على معلومات من قاعدة البيانات، تعتبر قراءة. مثلا معرفة المقاعد الفارغة في رحلة طيران أو معرفة الرصيد لحساب بنك هي عمليات قراءة، بينما سحب مبلغ من الرصيد أو حجز مقعد في رحلة طيران هي عمليات كتابة.

إذا تم تنفيذ عمليتي كشف حساب لمعرفة الرصيد من حساب مشترك، فلن يسبب هذا مشكلة، لأننا نريد القراءة من قاعدة البيانات (يسمح بأكثر من عملية قراءة في ذات الوقت). ولكن لا يسمح بتنفيذ أي عملية على البيانات المشتركة إذا كانت هنالك عملية كتابة تعمل على هذه البيانات، فإذا بدأت عملية في تعديل البيانات المشتركة فعلى كل العمليات الأخرى، بما فيها عمليات القراءة، التوقف والانتظار حتى تفرغ هذه العملية من عملها.

توفر نظم قواعد البيانات مثل أوراكل وأكسس إمكانية غلق السجلات التي نجري عليها التعديل (lock records) واستخدامها بصورة احتكارية (exclusively) حتى نمنع الوصول إليها من قبل أي عملية أخرى أثناء التعديل، وبهذا نضمن توافقية البيانات (consistency).

إذا كانت قاعدة البيانات مشتركة، فلابد لعملية الكتابة من الوصول للبيانات المشتركة بصورة احتكارية (exclusive access).

إذن استخدام الاحتكار يعتبر أحد الحلول لمشكلة القراءة والكتابة.

6.3.1.1. مشكلة القراءة والكتابة الأولى

The first readers-writers problem

إذا كانت هنالك عملية كتابة فلن نسمح لأي عملية أخرى بالوصول للبيانات المشتركة، ولكن إذا كانت العملية التي تتعامل مع البيانات عملية قراءة ، فيمكن لأي عملية قراءة أخرى من الوصول للبيانات المشتركة. أي أن عمليات القراءة تنفذ بمجرد وصولها، بينما توقف عملية الكتابة حتى لا يكون هنالك عملية قراءة.

6.5.1.2. مشكلة القراءة والكتابة الثانية

The second readers-writers problem

إذا كانت هنالك عملية قراءة ، ثم جاءت عملية كتابة، ستنتظر الأخيرة حتى تفرغ عملية القراءة، في هذه الاثناء إذا جاءت عملية قراءة أخرى (مسموح لعمليات القراءة أن تعمل مع بعضها)، ولكن ليس من اللائق أن تتخطى هذه العملية عملية الكتابة التي طلبت قبلها استخدام البيانات المشتركة. وإذا سمحنا لعملية القراءة الثانية بالعمل فقد تأتي عملية قراءة ثالثة ورابعة وخامسة وهكذا ، مما يتسبب في حرمان عملية الكتابة من العمل (starvation)، لذلك يجب على عملية الكتابة إذا وصلت صف الانتظار ألا تنتظر أكثر من اللازم وأن تبدأ العمل بمجرد وصولها. أي أن عمليات الكتابة تنفذ بمجرد وصولها، بينما تنتظر عمليات القراءة حتى لا تكون هنالك عملية كتابة.

6.5.1.3. مشكلة القراءة والكتابة الثالثة

The third readers-writers problem

مشكلة القراءة والكتابة الأولى تتسبب في حرمان عمليات الكتابة، لأن عمليات القراءة قد تأتي تباعا وتظل عملية الكتابة في انتظار دائم لا ينتهي.

مشكلة القراءة والكتابة الثانية تتسبب في حرمان عمليات القراءة لأن عمليات الكتابة عندما تأتي لصف الانتظار يسمح لها بالعمل عند أول فرصة لها، بالتالي قد تأتي عمليات الكتابة تباعا وتستأثر بالوصول للبيانات دون عمليات القراءة.

الحل الثالث جاء ليقول أنه يمنع احتكار الوصول للبيانات إذا تعدى فترة زمنية محددة، ولكل عملية قيد زمني في استخدامها للبيانات لا تتجاوزه، فإذا كانت هنالك عملية تستخدم البيانات احتكاريا فعليها أن تترك هذا الاحتكار وتنهي عملها إذا تعدى الفترة المقررة للعملية.

6.5.2. مشكلة المنتج والمستهلك (**producer-consumer**)

مشكلة المنتج والمستهلك أحيانا تسمى مشكلة الخازن المحدود (bounded-buffer). هنا يكون لدينا عمليتان تستخدمان خازن مشترك، العملية الأولى تضع به بيانات بينما تأخذ العملية الأخرى البيانات منه. وعلى العمليتين التنسيق فيما بينهما بحيث لاتحاول العملية الأولى وضع بيانات في الخازن إذا كان ممتلئاً، ولا تحاول العملية الثانية أخذ بيانات من الخازن إذا كان فارغاً.



شكل رقم (6-1): المنتج والمستهلك.

6.5.2.1. حل مشكلة المنتج والمستهلك

يمكن حل هذه المشكلة بإجبار عملية المنتج على التوقف عن إحضار البيانات للخازن إذا كان ممتلئا (sleep). وعندما يفرغ الخازن تقوم عملية المستهلك من إيقاظ المنتج ليبدأ في تعبئة الخازن. بنفس الطريقة تتوقف عملية المستهلك (sleep) إذا كان الخازن فارغا، وعندما يحضر المنتج بيانات للخازن، يقوم بإيقاظها لتبدأ في أخذ البيانات من الخازن. يمكن تطبيق هذا الحل باستخدام السيمافور (semaphore) الذي يعتمد على الاتصال بين العمليتين (inter-process communication).

6.5.2.1.1. حل يقود إلى اختناق

البرنامج (6-1) يعتبر حل للمشكلة، حيث يمثل count عدد العناصر الموجودة بالخازن، و المتغير item يمثل الخازن. هذا البرنامج يعمل كما يلي:

ستختبر عملية المنتج الخازن، هل هو ممتلئ أم لا ؟ وذلك بالأمر:

```
if (count == BUFFER_SIZE)
```

إذا كان الأمر صحيحا سيتوقف المنتج عن تعبئة الخازن ويذهب لينام (sleep)، ولن يرجع لعمله ما لم يوقظه المستهلك، أما إذا كان الخازن غير ممتلئ فسيواصل المنتج في إحضار عناصر للخازن بالأمر:

```
putItemIntoBuffer(item)
```

ثم يزيد count بواحد كما يلي:

```
count = count + 1
```

المستهلك سيختبر هل الخازن فارغ أم لا؟ بالأمر:

```
if (count == 0)
```

إذا كان الأمر صحيحا سيتوقف عن أخذ البيانات من الخازن ويذهب لينام ولن يرجع لعمله ما لم يوقظه المنتج (عندما يحضر بيانات للخازن). أما إذا كانت هنالك بيانات بالخازن فسيواصل المستهلك عمله بأخذ عنصر كل مرة من الخازن بالأمر:

```
item = removeItemFromBuffer()
```

ثم ينقص count بالأمر:

```
count = count – 1
```

البرنامج (6-1)

```
int count
procedure producer() {
```

```
    while (true) {
        item = produceItem()
        if (count == BUFFER_SIZE) {
            sleep()
        }
        putItemIntoBuffer(item)
        count = count + 1
       if (count == 1) {
            wakeup(consumer)
        }
    }
}
procedure consumer() {
    while (true) {
        if (count == 0) {
            sleep()
        }
        item = removeItemFromBuffer()
        count = count - 1
        if (count == BUFFER_SIZE - 1) {
```

```
        wakeup(producer)
      }
      consumeItem(item)
    }
}
```

هذا الحل سيقودنا إلى اختناق. لمعرفة كيف يحدث الاختناق دعنا نفترض السيناريو التالي:

- اختبر المستهلك count فوجده يحتوي على صفر ((if (count == 0)، لذلك سيذهب لينام (sleep).
- ولكنه قبل أن يصل فراشه تمت مقاطعته (قام نظام التشغيل بتوقيف المستهلك قبل تنفيذ sleep).
- بعدها سيضيف المنتج عنصر جديد إلى الخازن (أصبحت count تحتوي 1).
- يحاول المنتج إيقاظ المستهلك، وبما أن المنتج أصلا مستيقظ فإن طلب الإيقاظ هذا لافائدة منه (سيفقد).
- ولأن المستهلك كان قد اختبر الخازن (قبل إيقافه) ووجده فارغا، فإنه بعد أن يتم تشغيله مرة أخرى سيواصل من آخر نقطة كان يعمل فيها وهي الذهاب للنوم (استدعاء sleep)، ولن يستيقظ مرة أخرى، لأن أمر الإيقاظ (wakeup) قد فقد (قام به المنتج عند ما كان count=1).
- فيذهب المستهلك لينام ولن يستيقظ مرة أخرى (فهو لا يعلم أن أمر الإيقاظ قد فقد).
- سيستمر المنتج في عمله بإحضار عناصر جديدة إلى الخازن حتى يمتلئ:

```
if (count == BUFFER_SIZE)
```

- بعدها سيذهب لينام بافتراض أن المستهلك سيوقظه مرة أخرى، ولكن هذا لن يحدث.
- وبما أن العمليتان ستكونان في الفراش، وكل عملية غارقة في نومها وتنتظر الأخرى لتوقظها (ولا يوجد منبه)، فستظلان نائمتان هكذا إلى الأبد مما ينتج عنه اختناق لا محالة.

6.5.2.1.2. الحل باستخدام السيمافور (semaphore)

لحل مثل هذه المشكلة يمكننا استخدام ما يسمى بالسيمافور. السيمافور هو علامة كانت تستخدم قديما لتنظيم مرور القاطرات، حيث سيكون للسيمافور حالتين، إما تحت أو وفوق، عندما تصل قاطرة وتريد الدخول للمحطة سينظر السائق للسيمافور فإذا كانت وضعيته تحت فهذا يعني أن الطريق سالكة (تشبه إشارة المرور الخضراء)، ولأن خط السكة حديد لا يحتمل تلاقي قاطرتين، فسيرفع السيمافور إلى أعلى بعد دخول القاطرة بحيث لا يسمح لقاطرة أخرى أن تدخل (الخط مشغول) ويشبه هذا إشارة المرور الحمراء.

بنفس المفهوم سنستخدم متغير يمثل السيمافور (عدد صحيح يحتمل قيمتين)، حيث تمثل قيمة علامة تحت وقيمة أخرى علامة فوق. ثم إذا بدأت عملية استخدام بيانات مشتركة ستضع القيمة التي تمثل علامة فوق داخل متغير السيمافور ، وإذا أرادت عملية أخرى استخدام البيانات المشتركة ستختبر السيمافور وبما أن علامته فوق فهذا يعني أنه لا يسمح لها باستخدام البيانات المشتركة الآن.

دائما تختبر أي عملية السيمافور قبل الوصول للبيانات المشتركة واعتمادا على وضعه تتخذ العملية الإجراء اللازم. وعلى أي عملية تستخدم البيانات المشتركة أن تغير وضع السيمافور إلى فوق (ممنوع الاقتراب أو التصوير)، بعد أن تفرغ من البيانات المشتركة ستغير وضع السيمافور إلى تحت (تصبح البيانات متاحة للعمليات الأخرى). أهم خاصية في نظام السيمافور هو أنه إذا وضعت العملية قيمة فوق في السيمافور فلا يمكن لعملية أخرى الوصول إلى السيمافور حتى تكتمل العملية أو يتم توقيفها (حجزها).

في البرنامج (6-2) سنستخدم السيمافور fillCount والسيمافور emptyCount لتجنب تجاهل طلبات الإيقاظ ولحل مشكلة الاختناق.

في هذا البرنامج يغلق المنتج السيمافور emptyCount، (لا يستطيع المستهلك تفريغ الخازن في هذه اللحظة)، ثم بعد وضع عنصر في الخازن يفتح السيمافور fillCount.

عندما يريد المستهلك أخذ عنصر من الخازن سيغلق السيمافور fillCount، بحيث لا يستطيع المنتج التعبئة في هذه اللحظة، ثم بعد أخذ عنصر من الخازن يفتح السيمافور emptyCount.

برنامج رقم (6-2)

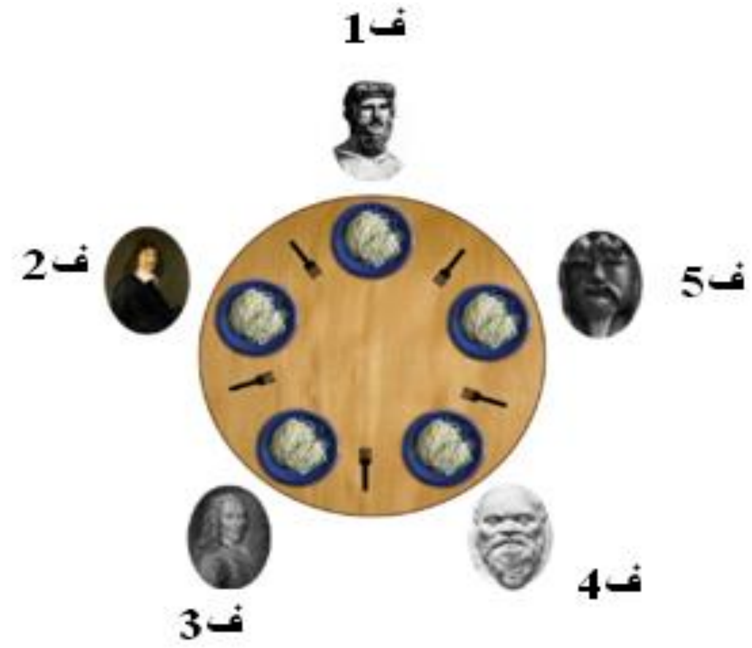
```
Semaphore fillCount=0
semaphore emptyCount = BUFFER_SIZE
procedure producer() {
    while (true) {
        item = produceItem()
        down(emptyCount)
        putItemIntoBuffer(item)
        up(fillCount)
    }
 }
procedure consumer() {
    while (true) {
        down(fillCount)
```

```
item = removeItemFromBuffer()
up(emptyCount)
consumeItem(item)
}
```

حل السيمافور مناسب إذا كان لدينا منتج واحد ومستهلك واحد. ولكن إذا كان هنالك أكثر من منتج أو/و أكثر من مستهلك فإن هذا الحل قد ينتج عنه (race condition).

6.5.3. مشكلة عشاء الفلاسفة (**Dining philosophers problem**)

هي مشكلة قديمة في التوازي، وهي عبارة عن مشكلة عمليات متعددة غير متزامنة. في عام 1971 ، أعد العالم ديكاسترا (Edsger Dijkstra) سؤال عن التزامن بين العمليات وهو يتحدث عن 5 حاسبات تتنازع على 5 سواقات أشرطة ممغنطة مشتركة. وبعدها بفترة قليلة أسماها العالم توني هوار (Tony Hoare) مشكلة عشاء الفلاسفة. وهي تمثيل نظري لعملية الاختناق والحرمان، حيث يقوم كل فيلسوف بأخذ شوكة واحدة في البداية ثم يبحث عن الشوكة الأخرى.



شكل رقم (6-2): عشاء الفلاسفة.

تعريف المشكلة

هنالك 5 فلاسفة يجلسون على طاولة مستديرة وعلى كل فيلسوف أن يقوم بأحد أمرين، إما أن يأكل أو يفكر. فإذا أكل لا يفكر، وإذا فكر لا يأكل.وهنالك خمسة شوك، شوكة بين كل فيلسوفين، بحيث يكون على يمين كل فيلسوف شوكة وعلى يساره شوكة. ويحتاج كل فيلسوف أن يستخدم الشوكتين للأكل. ولابد من أن يستخدم الشوكتين اللتين على يمينه ويساره مباشرة، الشكل (6-2). إذا أخذ كل الفلاسفة شوكاتهم اليسرى، عندها لن يستطيع أي منهم الحصول على الشوكة اليمنى، وسيظل كل فيلسوف منتظرا الشوكة اليمنى حتى تفرغ، أي أن الفيلسوف ف1، سيكون في انتظار الفيلسوف ف2 ليترك شوكته، والفيلسوف ف2 سينتظر الفيلسوف ف3 ليترك شوكته، وهكذا (انتظار دائري)، مما يسبب إختناق (deadlock).

حل غير مجدي:

يمكن حل المشكلة بجعل الفيلسوف يلتقط شوكته اليسرى ثم يختبر هل الشوكة اليمنى متاحة، فإن لم تكن كذلك يضع الفيلسوف شوكته اليسرى وينتظر فترة زمنية

محددة ثم يعيد الكرة مرة أخرى. هذه الطريقة قد تفشل إذا قام كل الفلاسفة بإلتقاط شوكاتهم اليسرى في وقت واحد، فلن يجدوا شوكاتهم اليمنى، فسيضعوا شوكاتهم اليسرى وينتظروا فترة زمنية محددة متساوية لكل الفلاسفة، ثم يلتقطوا شوكاتهم اليسرى مرة ثانية، ولن يجدوا اليمنى لذلك سيضعوا شوكاتهم اليسرى مرة أخرى وينتظروا فترة زمنية متساوية، وهكذا يظلوا يكررون في هذه المحاولة إلى ما لا نهاية وتظل المشكلة قائمة. هذا الأمر يحرم جميع الفلاسفة من الأكل (وتحصل مجاعة) أو ما يسمى الحرمان (starvation).

يمكن حل هذه المشكلة بجعل فترات انتظار الفلاسفة عشوائية وبالتالي إحتمال أنها تكون متساوية قد يكون ضعيفا جدا، وهذا ما يطبق بالفعل فى كثير من الأنظمة، لكن أحيانا نكون بحاجة إلى حل لا يحتمل الفشل بسبب تطابق الفترات العشوائية. ذلك لأن هنالك من الانظمة ما تدير أجهزة حساسة لا تقبل إحتمال أي فشل مثل أنظمة مراقبة المفاعل النووية.

حل آخر:

هنا نقوم باستخدام سيمافور ثنائي (semaphore)، حيث يقوم الفيلسوف قبل الحصول على شوكة بغلق السيمافور (تحت)، ثم بعد إعادة الشوكة يفتح السيمافور (فوق). هذه الطريقة تمكن فيلسوف واحد فقط أن يأكل في أي لحظة زمنية معينة، ولكن بما أن هنالك خمس شوك فمن المفترض أن يكون هنالك فيلسوفين قادرين على الأكل في اللحظة الواحدة.

حل ثالث:

هنا نتتبع حالة الفيلسوف هل هو:

- يأكل.
- يفكر.
- جائع (يريد أن يحصل على شوكة).

بحيث يستطيع الفيلسوف الجائع أن يأكل فقط إذا كان جاريه لا يأكلان.

6.5.4. مشكلة مدخني السجاير (Cigarette smokers problem)

هي أحد مشاكل التوازي التي تتحدث عن التزامن بين عن أربعة برامج. ظهرت في 1971. وقصتها كتالي:

إفترض أن السيجارة تتكون من ثلاث أجزاء هي:

- تبغ Tobacco.
- ورق paper.
- عود ثقاب match.

فإذا كان هنالك ثلاث مدخنين يجلسون حول طاولة، كل مدخن لديه مخزون كافي من أحد أجزاء السيجارة. وهنالك شخص رابع غير مدخن مهمته اختيار أثنين من المدخين الثلاثة عشوائيا، ليضع كل واحد من هؤلاء الإثنين قطعة واحدة من مما يمتلك من مخزونه على الطاولة، فمثلا إذا اخترنا صاحب التبغ وصاحب الورق، فسيضع كل منهما ما يكفي لعمل سيجارة واحدة. ويطلب من المدخن الثالث عمل السيجارة فيقوم باخذ ما في الطاولة ثم إضافة المكون الذي ليده لعمل السيجارة، مثلا إذا كان المدخن الثالث صاحب الثقاب، فسياخذ الورقة ويلف عليها التبغ الموجودين بالطاولة ثم يشعل السيجارة بعود ثقاب من عنده. عند ما تصبح الطاولة فارغة يقوم الشخص الرابع باختيار أثنين من المدخنين عشوائيا لتتم نفس الخطوات السابقة. ويظل تكرار هذه الخطوات دون توقف. لا يتم تخزين سجاير وانما تصنع السيجارة ثم تدخن مباشرة، ثم تصنع أخرى وتدخن وهكذا، فليس من المسموح به عمل أكثر من سيجارة في نفس الوقت. إذا تم وضع ورق وتبغ على الطاولة وكان صاحب اعواد الثقاب يدخن ستظل هذه المكونات على الطاولة دون ان يلمسها احد حتى يكمل صاحب الثقاب التدخين ثم يقوم بعمل السيجارة.

المشكلة تحاكي باربعة برامج تمثل الادوار الاربعة (ثلاث مدخنين و شخص رابع للاختيار). مسموح باستخدام السيمافور للتزامن، وممنوع لأي واحد من البرامج الاربعة بعمل قفز شرطي، فقط يمكن استخدام الانتظار المشروط باستخدام wait.

الحل

- تحل هذه المشكلة باستخدام سيمافرو ثنائي أو mutexes.
- نعرف مصفوفة من السيمافورات الثنائية A واحدة لكل مدخن.
- سيمافور ثنائي للطاولة T.
- هيئ كل سيمافورات المدخنين بقيمة إبتدائية صفر.
- وأجعل قيمة سيمافور الطاولة يبدأ بواحد.

.ستكون شفرة الشخص الذي يختار المدخنين هي:

```
while true{
        wait(T)
        choose smokers i and j nondeterministically
        making the third smoker k
        signal(A[k])
        }
```

وشفرة المدخن i هي:

```
while true {
          wait(A[i])
          make a cigarette
          signal(T)
```

smoke the cigarette

}

6.5.5. اللقاء Rendezvous

نفترض أن شابين قد تواعدا على أن يلتقيا في منتزه لم يرياه من قبل، بالفعل قد حضرا كل واحد على حدا، ولكنهما إندهشا من كبر حجم المنتزه، وبالتالي صعوبة أن يجد أحدهما الآخر. هنا على كل شاب أن يختار واحد من أمرين:

- أن ينتظر في مكان واحد ويتوقع أن الآخر سيبحث عنه.
- أن يبدأ بالبحث بإفتراض أن الآخر سينتظر في مكان واحد.

السؤال هنا أي استراتيجية يجب أن يختارا حتى يكون إحتمال اللقاء أكبر ؟

- إذا قرر الإثنين الإنتظار فبالتأكيد لن يجد أحدهما الآخر أبدا.
- إن قررا أن يبحثا عن بعضهما فهنالك فرصة في ان يتقابلا.
- إذا قرر أحدهما أن ينتظر والآخر أن يبحث فحتما سيلتقيا (من ناحية نظرية).

6.5.6. مشكلة الحلاق النائم (sleeping barber)

إذا افترضنا أن لدينا صالون حلاقة به حلاق واحد وكرسي حلاقة واحد وعدد من الكراسي للإنتظار. إذا لم يكن هنالك زبائن سيجلس الحلاق في كرسي الحلاقة وينام، وعندما يصل زبون فسيقوم بإيقاظ الحلاق. إذا جاء زبون آخر وكان الحلاق يحلق للزبون الأول فسيجلس الزبون الثاني على أحد كراسي الإنتظار. وكلما يصل زبون سيجلس في كرسي إنتظار إلى أن تمتلي كراسي الإنتظار. إذا جاء زبون ووجد كراسي الإنتظار مشغولة فعليه مغادرة الصالون.

الحل

لحل هذه المشكلة نستخدم ثلاث سيمافورات:

- سيمافور للزبائن المنتظرين

- سيمافور للحلاق (لنعرف هل هو نائم (عاطل) أم يعمل)
- سيمافور لتحقيق والتأكد من المنع المتبادل (mutual exclusion): بحيث لا نسمح لشخصين بالحلاقة في وقت واحد.

كل ما يحضر زبون سيحاول الحصول على كرسي الحلاقة (mutex) وسيظل كذلك حتى ينجح. على الزبون الذي يحضر للصالون أن يحسب عدد الزبائن المنتظرين فإذا كان أقل من عدد الكراسي فسيجلس وإلا فسيغادر (يحاول الحصول على كرسي سواء في غرفة الإنتظار أو كرسي الحلاق نفسه).

إذا وجد الزبون كرسي سيجلس وينقص عدد الكراسي الفارغة بواحد (critical section).

يقوم الزبون بإسال إشارة إلى الحلاق لإيقاظه، وسيحرر mutex للسماح للزبائن (أو الحلاق) بالمقدرة على الحصول عليه. إذا كان الحلاق مشغول فعلى الزبائن الانتظار. سيجل الحلاق في إنتظار دائم، يتم ايقاظه بأي زبون من المنتظرين، وعندما يستيقظ سيرسل إشارات للزبائن المنتظرين بواسطة السيمافور للسماح لهم بالحلاقة ، واحد كل مرة. هذه المشكلة تحتوي على حلاق واحد لذلك تسمى أحيانا (*single sleeping barber problem*). فيما يلي شفرات مبدئية (pseudo-code)،تضمن التزامن بين الحلاق والزبائن خالية من الاختناق، ولكنها قد تقود إلى حرمان الزبون.

نجد P و V هما دالتين توفرهما السيمافورات.

```
Semaphore Customers = 0
Semaphore Barber = 0
Semaphore accessSeats (mutex) = 1
int NumberOfFreeSeats = N // عدد المقاعد الفارغة
```

عملية أو خيط الحلاق:

```
while(true) { // تكرار غير منتهي
P(Customers) // محاولة الحصول على زبون وإذا لايوجد زبائن ينام
P(accessSeats) //  في هذه اللحظة تم إيقاظه، يريد تعديل عدد
المقاعد المتاحة
NumberOfFreeSeats++ //لقد أصبح هنالك مقعد فارغ
V(Barber)  //الحلاق جاهز ليبدأ الحلاقة
V(accessSeats) // لانريد إغلاق الكراسي
   // الحلاق الآن يحلق لزبون
```

```
}
```

عملية أو خيط الزبون:

```
 while(true) { //تكرار لا منتهي
   P(accessSeats) // محاولة الحصول على مقعد
   if ( NumberOfFreeSeats > 0 ) { // إذا كان هنالك مقعد فارغ
     NumberOfFreeSeats-- //   إجلس على المقعد
     V(Customers) // أخبر الحلاق ، الذي هو في انتظار زبون
     V(accessSeats) // لا نحتاج إلى إغلاق الكراسي
     P(Barber) //  الآن حان دور الزبون، لكنه سينتظر إذا كان
الحلاق مشغول
     // هنا سيتمكن الزون من الحلاقة
   } else { // لا توجد مقاعد فارغة
     V(accessSeats) // حرر إغلاق المقاعد
     //سيغادر الزيون دون أن يحلق
   }
 }
```

6.6. تمارين غير محلولة

1. ما هو مفهوم التزامن ؟
2. ما المقصود بالنزاع ؟
3. ما الفرق بين العملية المستقلة والعملية المتعاونة ؟
4. لماذا تحتاج العمليات المتوازية التعاون فيما بينها ؟
5. ما المقصود بحالة السباق (race condition) ؟ وما هي مسبباتها ؟ وكيف يتم حلها ؟
6. عرف المنطقة الحرجة (crtitical section) ؟
7. عرف اللقاء Rendezvous ؟
8. أذكر خمسة من مشاكل التزامن الكلاسكية ؟
9. تتصل العمليات المتعاونة فيما بينها بطريقتين، ما هما ؟

الباب السابع:

الاختناق

الباب السابع

الاختناق (deadlock)

في البرمجة المتعددة تتنافس العمليات على الموارد، وقد تطلب العملية مورد ما، إذا لم يكن متاح في ذلك الوقت، ستضطر العملية لإنتظاره (تتحول إلى حالة الإنتظار)، وقد تظل العملية في هذه الحالة ولا تتغير منها أبدا، لأن عملية أخرى تحجز هذا المورد هي أيضا في حالة إنتظار . هذا الوضع نسميه إختناق

في هذا الباب سنتحدث عن كيف يتعامل نظام التشغيل الإختناق.

7.1. تعريف المورد (*resource*)

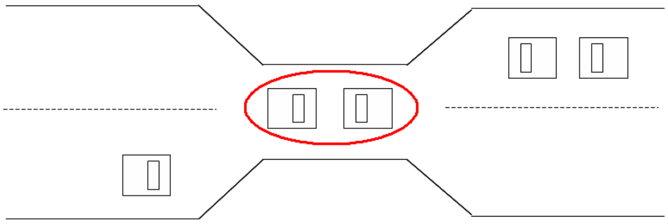
المورد هو كل ما يستخدمه البرنامج من عتاد (مثل الطابعة، القرص الصلب، والذاكرة) وبرامج وملفات (مثل جدول قاعدة بيانات، برنامج ما، صفحة إنترنت).

7.2. مفهوم الاختناق

إذا كان لدينا عمليتين (أ و ب)، وكانت العملية أ تستخدم عدة موارد (من بينها المورد م1) وتحتاج إلى مورد إضافي هو المورد م2، في نفس الوقت لدينا العملية ب التي تستخدم المورد م2 وتحتاج إلى مورد آخر لتكمل تنفيذها هو المورد م1 (الذي بحوذة العملية أ)، هنا ستظل العمليتان في هذا الوضع إلى ما شاء الله، فكل عملية لن تكتمل لأنها تحتاج مورد العملية الأخرى، هنا يحدث الاختناق لأن كل عملية تحتكر مواردها التي تستخدمها ولا تتركها أبدا.

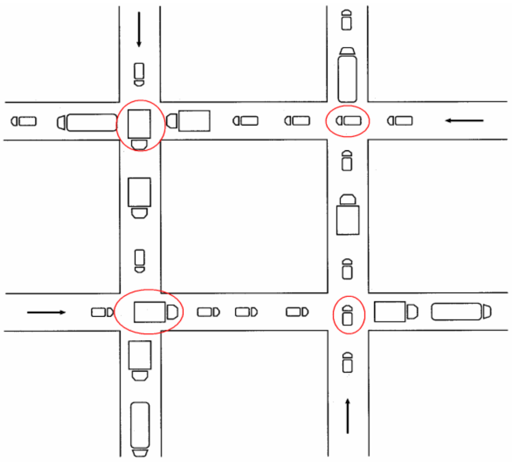
7.2.1. مثال توضيحي

إذا كنت تقود سيارة (عملية) ثم دخلت في ممر ضيق (مورد) يتسع لسيارة واحدة فقط، وجاءت سيارة من الاتجاه المعاكس (عملية أخرى) ودخلت نفس الممر ، فالتقت السيارتان في منتصف الممر وكل سيارة تريد المرور عبر الممر، هنا نقول أن هنالك اختناق في الممر، شكل (7-1).



شكل رقم (7-1): اختناق داخل ممر.

مثال آخر، إذا كان هنالك تقاطعات طرق كما في الشكل (7-2)، فإن أي سيارة لا يمكنها التحرك ما لم نخرج بعض السيارات خارج الطريق.



شكل رقم (7-2): اختناق سيارات في تقاطعات طرق.

7.2.2. معالجة الاختناق

نلاحظ أن الاختناق دوما يمكن حله، فمثلا في اختناق الممر يمكن لسيارة أن ترجع للخلف لتترك السيارة الأخرى لتمر (إجبار سيارة على التخلي عن الممر).

في مثال تقاطع الطرق نجد أن إزاحة بعض السيارات خارج الطريق يحل الاختناق.

7.3. أنواع الموارد

تنقسم الموارد إلى نوعين هما:

- موارد قابلة للنزع (preemptable resources): يمكن نزع هذا النوع من الموارد من العملية التي تستخدمه دون أن يسبب ذلك مشاكل، مثل الذاكرة.
- موارد غير قابلة للنزع (non-preemptable resources): قد يحدث خلل بالعملية (تتعطل) إذا انتزعنا منها المورد، مثل الطابعة، مسجل الأسطوانات الضوئية (CD writer).

عندما تريد عملية استخدام مورد فإنها تقوم بطلب المورد أولا، ثم تستخدمه، وعند الانتهاء منه تتخلى عنه لتستفيد منه عملية أخرى.

7.4. مسببات الاختناق

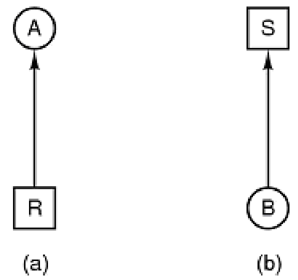
هنالك عدة أسباب إذا تحققت في مجموعة عمليات سيحدث اختناق لا محالة، هذه الأسباب الأربع هي:

1. المنع المتبادل (*mutual exclusion*): عملية واحدة فقط هي التي تستخدم مورد معين في اللحظة المعينة فلا يمكن لعمليتين استخدام نفس المورد في نفس الوقت.
2. الاحتفاظ والانتظار (*hold and wait*): تحتفظ العملية بمواردها التي تستخدم وتريد موارد أخرى لتكمل عملها.
3. الاحتكار أو عدم التخلي عن المورد (non preemption) : لا يمكن انتزاع المورد من العملية التي تستخدمه ما لم يكتمل عملها.
4. الانتظار الدائري (circular wait): يحدث الانتظار الدائري في سلسلة من العمليات إذا كانت العملية الأولى بالمجموعة تستخدم موارد معينة وتريد موارد أخرى من العملية الثانية في المجموعة، والعملية الثانية تستخدم الموارد التي تريدها العملية الأولى لأنها لم تفرغ منها بعد، ولتتخلى عن مواردها التي تستخدم تحتاج موارد أخرى تستخدمها العملية الثالثة ، وهكذا إلى العملية الأخيرة في المجموعة والتي تستخدم موارد تريدها العملية قبل الأخيرة، وتحتاج موارد تستخدمها العملية الأولى مما يشكل دائرة من الانتظار.

7.5. استخدام الرسومات

يمكننـا اسـتخدام بعـض الرسـومات لتوضـيح مسـببات الاختنـاق. مـثلا يمكـن استخدام:

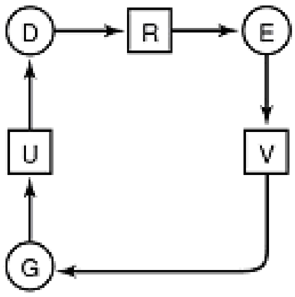
- المربع ليدل على مورد، في الشكل (7-3)، يعتبر كل من R و S موارد.
- الدائرة لتدل على عملية، في الشكل (7-3)، يعتبر كل من A و B عمليات.
- السهم الذي يخرج من دائرة (عمليـة) ويشير إلى مربع (مورد) يعني أن العملية تريـد (تطلـب هذا المورد)، مـثلا فـي الشـكل (7-3) – (b)، نجد أن العمليـة B تريد المورد S.
- السهم الذي يخرج مـن مربـع (مـورد) ويتجـه إلـى دائـرة (عمليـة) يعنـي أن هـذا المورد يستخدم بواسطة العملية، في الشكل (7-3) – (a)، المـورد R مستخدم بواسطة العملية A.



شكل رقم (7-3): رسومات توضح العلاقة بين الموارد والعمليات.

7.5.1. تمثيل الانتظار الدائري بالرسم

مثلا في الشكل (7-4)، نجد أن هنالك انتظـار دائـري حيث نجد أن العمليـة D تستخدم المورد U وتحتاج إلى المورد R، ونجد أن المـورد R تستخدمه العمليـة E التـي تريـد المـورد V الـذي يستخدم بواسطة العمليـة G، ونجـد أن G تريـد المـورد U الـذي تستخدم العملية D وهكذا نجد أنفسنا في دائرة انتظار.



شكل رقم (7-4): انتظار دائري.

إذا ابتدأت من أي عملية أو مورد في أي شكل ، وتتبعت اتجاه الأسهم فقادتك إلى نفس النقطة التي بدأت منها فأنت في انتظار دائري بلا شك، مثلا في الشكل (7-4)، إذا بدأت من أي نقطة (R) مثلا وتحركت مع اتجاه الاسم ستصل مرة أخرى إلى R مما يدل على وجود انتظار دائري.

7.6. التعامل مع الاختناق

هل تريد وقاية نفسك من الاختناق أم تريد إصلاحه بعد حدوثه (العلاج) ؟ المثل يقول الوقاية خير من العلاج، لذلك الأفضل أن نقوم باستخدام الموارد وحجزها بصورة حذرة حتى لا نقع في الاختناق (نقي أنفسنا من حدوثه)، ولكن إذا حدث بالرغم من حذرنا فعلينا معالجته أو تجاهله، هكذا يقول علماء نظم التشغيل.

يمكن التعامل مع الاختناق بطريقتين :

- الوقاية وذلك بمنع حدوثه أو تجنبه.
- العلاج وذلك بتجاهله أو إصلاحه، طبعا التجاهل لا يحتاج كثير عناء ، فكل ما على نظام التشغيل هو التظاهر بعدم حدوثه (وكأن شي لم يحدث). أما الإصلاح فيحتاج مرحلة قبله هي اكتشاف الاختناق أولا، ثم إصلاحه ثانيا.

7.6.1. تجاهل الاختناق (خوارزمية النعامة (*The Ostrich Algorithm*))

أبسط خوارزمية تستخدم في تجاهل الاختناق هي خوارزمية النعامة (أدفن رأسك في الرمل وتظاهر بأنه لا توجد مشكلة).

التظاهر بعدم حدوث الاختناق أو الهروب من المشكلة نلجأ إليها لسببين هما:

- إذا كان الاختناق يحدث نادرا.
- إذا كان علاجه مكلّفا.

7.6.2. اكتشاف الاختناق

هنالك طرق عديدة لاكتشاف الاختناق.

7.6.2.1. اكتشاف الاختناق لمورد واحد من كل نوع

هنا سنستخدم الرسم للاكتشاف الاختناق في نظام به مورد من كل نوع (مثلا طابعة واحدة، ماسحة واحدة، ..الخ).

مثال

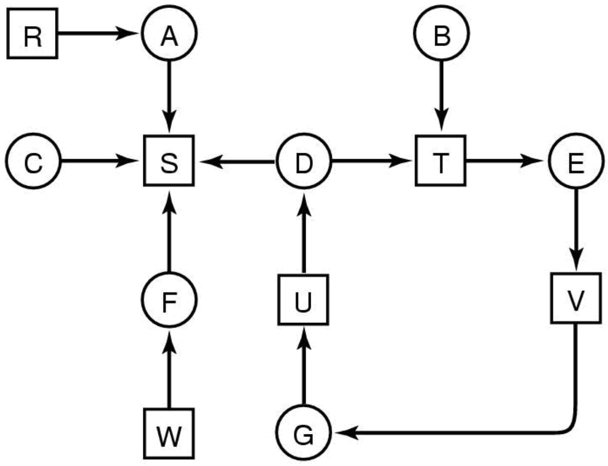
إذا كان لدينا 7 عمليات من A إلى G و مورد واحد من كل نوع، وكانت طلبات العمليات كما يلي:

- العملية A تستخدم المورد R وتريد المورد S لإتمام عملها.
- العملية B لا تستخدم أي مورد وتريد المورد T لإتمام عملها.
- العملية C لا تستخدم أي مورد وتريد المورد S لإتمام عملها.
- العملية D تستخدم المورد U وتريد المورد S و المورد T لإتمام عملها.
- العملية E تستخدم المورد T وتريد المورد V لإتمام عملها.
- العملية F تستخدم المورد W وتريد المورد S لإتمام عملها.
- العملية G تستخدم المورد V وتريد المورد U لإتمام عملها.

نريد أن نعرف هل سيكون هنالك اختناق أم لا (اكتشاف الاختناق) ؟

الحل

سنقوم بإنشاء رسم لكل الموارد مع العمليات التي تستخدمها والتي تريدها، فيكون الرسم كما في الشكل (7-5).



شكل رقم (7-5): رسم يبين الموارد.

نلاحظ من الشكل (7-5)، أن هنالك اختناق، لأنني مثلا إذا بدأت من المورد U ثم تتبعت اتجاه الأسهم سأجد نفسي في U مرة أخرى وهذا يعني وجود انتظار دائري، مع وجود مسببات الاختناق الأخرى.

7.6.2.2. اكتشاف الاختناق لعدة موارد

قد يكون لدينا عدد من الموارد من كل نوع (مثلا 5 طابعات، 4 ماسحات، ...الخ)، في هذه الحالة يصعب استخدام الرسومات للتعبير عن الموارد المتعددة لذلك سنستخدم طريقة المحددات (matrix) لاكتشاف الاختناق.

إذا كان لدينا عدد n عملية و عدد m نوع من الموارد، حيث يمثل E1 عدد الموارد من النوع الأول (مثلا عدد الماسحات) ، E2 يمثل عدد الموارد من النوع الثاني (مثلا عدد الطابعات)، وهكذا إلى Em. بحيث ننشي متجه (vector) للموارد التي توجد بالنظام. مثلا إذا كان لدى 7 طابعات، 6 ماسحات، 5 أقراص، فإن المتجه الذي يمثل هذه الموارد سيكون كالتالي:

$$(5 \quad 6 \quad 7)$$

شكل رقم (7-6)

إذا كان لدينا عدد 5 طابعات، 4 ماسحات، و 3 أقراص مستخدمة بواسطة عمليات مختلفة، فإن المتجه الذي يمثل الموارد المستخدمة سيكون:

(3 4 5)

شكل رقم (7-7)

ومن هنا يمكننا إنشاء متجه يمثل الموارد المتاحة (غير المستخدمة) وذلك بطرح الموارد المستخدمة من الموارد الكلية كما يلي:

(7 6 5) = الموارد الكلية.

(5 4 3) = الموارد المستخدمة.

(2 2 2) = الموارد المتاحة

شكل رقم (7-8)

<u>محددة الاستخدام (الموارد التي تستخدمها العمليات حاليا)</u>

يمكننا تكوين محددة للموارد المستخدم بواسطة عدة عمليات، حيث يمثل كل صف في المحددة موارد عملية معينة. مثلا لو أردنا إنشاء محددة لأربع عمليات كانت تستخدم الموارد أعلاه، فإن المحددة ستكون كما يلي:

نوع المورد

	الطابعات	الماسحات	الأقراص
الموارد التي تستخدمها العملية الأولى	5	1	1

الموارد التي تستخدمها العملية الثانية	0	1	1
الموارد التي تستخدمها العملية الثالثة	0	1	0
الموارد التي تستخدمها العملية الرابعة	0	1	1

شكل رقم (7-9)

من الشكل (7-9) نجد أن:

- العملية الأولى تستخدم 5 طابعات، ماسحة، وقرص.
- العملية الثانية تستخدم ماسحة وقرص.
- العملية الثالثة تستخدم ماسحة واحدة فقط.
- العملية الرابعة ماسحة وقرص.

<u>محددة الطلبات (الموارد التي تحتاجها العمليات)</u>

بنفس الطريقة يمكن إنشاء محددة للموارد التي تحتاجها كل عملية من كل مورد، كما في المثال التالي:

	الطابعات	الماسحات	الأقراص
الموارد التي تحتاجها العملية الأولى	2	2	2
الموارد التي تحتاجها العملية الثانية	5	6	7
الموارد التي تحتاجها العملية الثالثة	4	3	3
الموارد التي تحتاجها العملية الرابعة	4	4	4

شكل رقم (7-10)

من الشكل (7-10) نجد أن:

- العملية الأولى تحتاج إلى طابعتين وماسحة وقرصين.

- العملية الثانية تحتاج إلى 5 طابعات، 6 ماسحات و7 أقراص.
- العملية الثالثة تحتاج 4 طابعات، 3 ماسحات، و3 أقراص.
- العملية الرابعة تحتاج إلى 4 طابعات، 4 ماسحات، و 4 أقراص.

اكتشاف الاختناق بالمحددات

الآن لدي عدد الموارد الكلية (شكل 7-6)، وعدد الموارد الحرة (شكل 7-8)، وعدد الموارد المستخدمة (شكل 7-9)، وعدد الموارد المطلوبة التي تحتاجها العمليات لإتمام عملها (شكل 7-10)، سنقوم باستخدام هذه المعطيات لاكتشاف هل سيحدث اختناق أم لا ؟

طريقة الحل

سنقوم أولا بمقارنة الموارد الحرة (شكل 7-8) مع صفوف محددة الموارد المطلوبة، ثم نختار الصف (العملية) الذي تكون موارده المطلوبة أقل أو تساوي الموارد الحرة المتوفرة. بمقارنة صفوف المحددة في الشكل (7-10) مع الموارد الحرة في الشكل (7-8) سنجد أن العملية الأولى يمكنها إكمال عملها باستخدام الموارد الحرة المتاحة. سنعطي العملية الأولى ما تحتاج من الموارد المتاحة ونخصم ذلك من الموارد الحرة كما يلي:

(2 2 2) = الموارد الحرة

(2 2 2) = الموارد التي تحتاجها العملية الأولى

(0 0 0) = المتبقي من الموارد الحرة.

شكل رقم (7-11)

ملحوظة

إذا كانت الموارد الحرة تكفي لواحد من عمليتين، فالأفضل اختيار العملية التي تستخدم موارد أكثر ذلك لأني سأحصل على موارد حرة أكثر بعد أن تكمل العملية عملها وتحرر ما تستخدم من موارد.

الآن ستكمل العملية الأولى عملها ثم تحرر ما لديها من موارد (التي كانت تستخدم مسبقا والتي إعطيناها لها في الخطوة السابقة)، بالتالي سيكون لدي من الموارد الحرة ما يلي:

(0 0 0) = الموارد الحرة المتبقية بعد إعطاء العملية الأولى ما تحتاج من موارد

(5 1 1) = الموارد التي كانت تستخدمها العملية الأولى من قبل (من محددة الاستخدام)

(2 2 2)= الموارد التي استخدمتها العملية الأولى مؤخرا (من محددة المطلوب)

(7 3 3) = الموارد الحرة بعد انتهاء العملية الأولى وتحريرها لمواردها

شكل رقم (7-12)

الآن سنقارن الموارد الحرة التي حصلنا عليها بعد انتهاء العملية الأولى وتحرير مواردها مع المتبقي من صفوف محددة المطلوب (شكل 7-10). سنجد أن العملية التي يمكن أن تكفيها الموارد الحرة هي العملية الثالثة، فنخصم الموارد التي تريد من الموارد الحرة التي لدينا كما في الشكل (7-13).

(7 3 3) = الموارد الحرة

(4 3 3) = الموارد التي تحتاجها العملية الثالثة

(3 0 0) = المتبقي من الموارد الحرة.

شكل رقم (7-13)

الآن ستكمل العملية الثالثة عملها ثم تحرر ما لديها من موارد (القديم والجديد)، بالتالي سيتوفر لدي من الموارد التالي (بعد انتهاء العملية الثالثة):

(3 0 0) = الموارد الحرة المتبقية بعد إعطاء العملية الأولى ما تحتاج من موارد

(0 1 0) = الموارد التي كانت تستخدمها العملية الثالثة من قبل (من محددة الاستخدام)

(4 3 3)= الموارد التي استخدمتها العملية الثالثة مؤخرا (من محددة المطلوب)

(7 4 3) = الموارد الحرة بعد انتهاء الثالثة وتحريرها لمواردها

شكل رقم (7-14)

الآن سنقارن الموارد الحرة التي حصلنا عليها بعد انتهاء العملية الثالثة وتحرير مواردها كما في الشكل (7-14)، مع المتبقي من العمليات التي بمحددة المطلوب (شكل 7-10). سنجد أن الموارد المتوفرة لدينا لا تكفي لأي عملية من العمليات المتبقية. وبالتالي يمكننا القول أن هنالك اختناق حدث في هذه النقطة.

مثال (2)

الآن سنعرض مثالا آخر بدون شرح مفصل يوضح حالة لا يحدث فيها اختناق.

المعطيات:

الموارد الموجودة بالنظام =(4 2 3 1)

المستخدم منها =

0	0	1	0
2	0	0	1
0	1	2	0

الموارد المطلوبة =

2	0	0	1
1	0	1	0
2	1	0	0

المطلوب: هل يوجد اختناق أم لا ؟

الحل

أولا نحسب الموارد المتوفرة وذلك بطرح الموارد المستخدمة (مجموع أعمدة محددة الموارد المستخدمة) من الموارد كما يلي:

موارد النظام	1	3	2	4
الموارد المستخدمة	1	3	1	2
الموارد المتوفرة	0	0	1	2

ثم نقارن المتاح مع صفوف المطلوب (في محددة المطلوب)، فنجد أن الصف الثالث يمكنه العمل ، فنعطيه ما يريد من موارد كما يلي:

الموارد المتوفرة =	0	0	1	2
الموارد المطلوبة (صف 3)	0	0	1	2
الموارد المتوفرة بعد إعطاء صف 3 موارده	0	0	0	0

بعد انتهاء العملية صف 3، سنستفيد من موارده، فيصبح لدينا الموارد المتاحة التالية:

الموارد المتوفرة بعد إعطاء صف 3 موارده	0	0	0	0
موارد صف3 في محددة المستخدم	0	2	1	0
موارد صف3 في محددة المطلوب	0	0	1	2
الموارد المتوفرة بعد انتهاء صف 3 من عمله	0	2	2	2

الموارد المطلوبة =

1	0	0	2
0	1	0	1
0	0	1	2

الآن سنقارن المتاح مع صفوف المطلوب (في محددة المطلوب)، فنجد أن الصف الثاني يمكنه العمل ، فنعطيه ما يريد من موارد كما يلي:

الموارد المتوفرة =	(2	2	2	0)
الموارد المطلوبة (صف 2)	(1	0	1	0)
الموارد المتوفرة بعد إعطاء صف 2 موارده	(1	2	1	0)

بعد انتهاء العملية صف 2، سنستفيد من موارده، فيصبح لدينا الموارد المتاحة التالية:

الموارد المتوفرة بعد إعطاء صف 2 موارده	(1	2	1	0)
موارد صف2 في محددة المستخدم	(2	0	0	1)
موارد صف2 في محددة المطلوب	(1	0	1	0)
الموارد المتوفرة بعد انتهاء صف 2 من عمله	(4	2	1	1)

الموارد المطلوبة =	2	0	0	1
	1	0	1	0
	2	1	0	0

نجد أن الصف المتبقي في محددة المطلوب تكفيه الموارد المتوفرة بعد انتهاء صف2، وهذا يعني أنه لا يوجد اختناق.

7.6.3. معالجة الاختناق

الخطوة السابقة وضحت كيف يمكننا اكتشاف الاختناق، هنا سنقوم بمعالجة الاختناق بعد إكتشافه، ويتم ذلك بعدة طرق منها:

- انتزاع بعض الموارد من أحد عمليات الإختناق: يجب التأكد من أن الموارد قابل للنزع، بحيث لا يسبب نزعه مشكلة للعملية، مثلا إذا كانت العملية تقوم بتسجيل بيانات على اسطوانة ضوئية (تستخدم مسجل الأسطوانات الضوئية

(CD writer) فإن انتزاع هذا المورد من العملية في هذه اللحظة قد يتسبب في تعطل الاسطوانة. كذلك انتزاع طابعة من عملية وهي تطبع قد يجعل مخرج الطابعة مبتور وغير واضح. لكن إذا كان الاختناق على الذاكرة فيمكن تحويل أحد العمليات المتسببة في الاختناق إلى القرص الصلب (انتزاع الذاكرة من العملية) ، وإرجاعها فيما بعد دون حدوث مشكلة.

- إيقاف بعض العمليات التي تشارك في الاختناق: يمكن توقيف عملية (قتلها كما يقال في لينكس (kill)) والاستفادة من مواردها لتشغيل بقية العمليات. في هذه الحالة سنحاول قتل العمليات التي يمكن تشغيلها من جديد بسهولة ودون أن تسبب مشاكل.

7.6.4. الوقاية

يمكننا أن نقي النظام من حدوث الاختناق وذلك بطريقتين:

- تجنبه
- منعه

7.6.4.1. تجنب الاختناق باستخدام خوارزمية المصرف Banker's Algorithms

هنا يحاول النظام التأكد من أن الموارد يمكن حجزها بصورة آمنة، وذلك قبل الشروع في حجزها. هنالك العديد من الخوارزميات التي يمكن استخدامها للتأكد من أن الموارد يمكن حجزها بدون أن يحدث اختناق (صورة آمنة safe state).

مثال

إذا كان لدينا موارد ، منها المستخدم ومنها غير المستخدم (الحر)، وهنالك عمليات تستخدم موارد وتريد المزيد، فيمكننا استخدام الطريقة التالية للتأكد من أن حجز

الموارد الحرة للعمليات التي تريد موارد لا يسبب اختناق، مثلا الشكل (7-14) يوضح ثلاث عمليات (أ، ب، ج) ، وكل عملية تستخدم عدد معين من الموارد وتحتاج أن تصل إلى عدد معين من الموارد لتعمل.

العملية	لديها	يكفيها
أ	3	9
ب	2	4
ج	2	7

الموارد المتوفرة : 3

شكل رقم (7-15): العمليات، الموارد المستخدم والمطلوبة.

العملية أ تستخدم 3 موارد وتحتاج أن تصل إلى 9 موارد، العملية ب لديها موردين وتريد أن تصل إلى 4 موارد، العملية ج لديها موردين وتريد أن تكمل مواردها إلى 7 موارد لتعمل. و لدينا ثلاث موارد متاحة. نريد التأكد من أننا لو استخدمنا هذه الموارد المتوفرة يمكن أن نصل إلى حالة آمنة (لا يحدث اختناق).

الحل

بمقارنة المتوفر مع المطلوب، نجد أن العملية ب يمكن أن تعمل وذلك بإعطائها موردين من المتوفر فيكون لدى مورد واحد متوفر كما في الشكل (7-16).

العملية	لديها	يكفيها
أ	3	9
ب	4	4
ج	2	7

الموارد المتوفرة: 1

شكل رقم (7-16)

بعد انتهاء العملية ب ستترك الموارد التي كانت تستخدم (4 موارد)، فيصبح المتوفر هو 5، كما في الشكل (7-17).

العملية	لديها	يكفيها
أ	3	9
ب	-	-
ج	2	7

الموارد المتوفرة: 5

شكل رقم (7-17)

بمقارنة المتوفر مع المطلوب، نجد أن العملية ج يمكن أن تعمل وذلك بإعطائها 5 موارد وهي كل ما يتوفر كما في الشكل (7-18).

العملية	لديها	يكفيها
أ	3	9
ب	-	-
ج	7	7

الموارد المتوفرة: 0

شكل رقم (7-18)

بعد انتهاء العملية ج ستترك الموارد التي كانت تستخدم (7 موارد)، فيصبح المتوفر هو 7، كما في الشكل (7-18).

العملية	لديها	يكفيها

أ	3	9
ب	-	-
ج	-	-

الموارد المتوفرة: 7

شكل رقم (7-18)

بمقارنة المتوفر مع المطلوب، نجد أن العملية أ يمكن أن تعمل وذلك بإعطائها 6 موارد ويبقى لي مورد واحد غير مستخدم كما في الشكل (7-19).

العملية	لديها	يكفيها
أ	9	9
ب	-	-
ج	-	-

الموارد المتوفرة: 1

شكل رقم (7-19)

بعد انتهاء العملية أ ستترك الموارد التي كانت تستخدم (9 موارد)، فيصبح المتوفر هو 10، كما في الشكل (7-20).

العملية	لديها	يكفيها
أ	-	-
ب	-	-
ج	-	-

الموارد المتوفرة: 10

شكل رقم (7-20)

هذا يدل أن الموارد يمكن استخدامها دون حدوث اختناق (حالة آمنة).

مثال

إذا لم نستخدم الموارد بحذر لتجنب الاختناق ، فقد نصل إلى حالة غير آمنة (unsafe) مما يتسبب في اختناق. المثال السابق يمكن استخدامه لتوضيح كيف يمكن أن يحدث اختناق كما في الشكل (7-21).

أ	3	9
ب	2	4
ج	2	7

الموارد المتوفرة : 3

أ	4	9
ب	2	4
ج	2	7

الموارد المتوفرة: 2

أ	4	9
ب	4	4
ج	2	7

الموارد المتوفرة: 0

أ	4	9
ب	-	-
ج	2	7

الموارد المتوفرة: 4

شكل رقم (7-21)

هنا وصلنا إلى حالة غير آمنة وحدث الاختناق، لان المتوفر (4 موارد) لا يكفي لأي عملية.

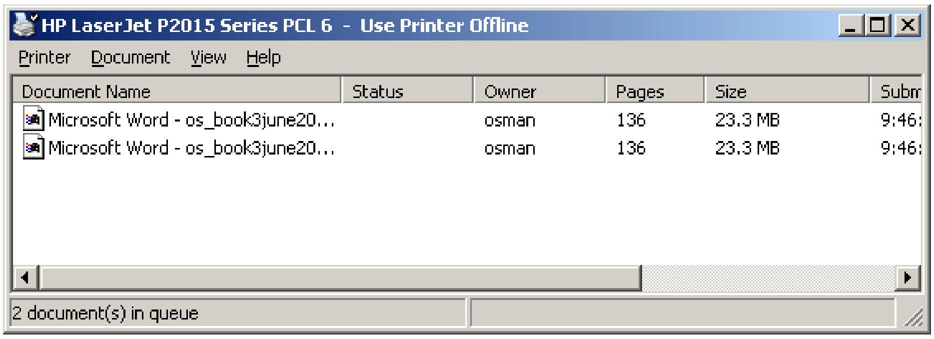
7.6.4.2. منع الاختناق

تجنب الاختناق قد يكون غير ممكن لأنه يتطلب دراية مسبقة عن الطلبات المستقبلية والتي لا يمكن معرفتها في غالبا الأحيان، لذلك سنجد أن الوقاية من الاختناق يمكن أن تتحقق بنفي واحد أو أكثر من مسببات الاختناق. فمن المعلوم أن مسببات الاختناق لابد من توافرها جميعها (أربعة مسببات) حتى يحدث اختناق، بالتالي إذا استطعنا نفي سبب واحد من هذه المسببات الأربع سنكون وقينا أنفسنا من الاختناق.

7.6.4.2.1. نفي المنع المتبادل

إذا كان لدى مورد لا تقبل المشاركة ولا يمكن استخدام بأكثر من عملية في نفس الوقت، يمكنني تجنب الاحتكار وذلك بتخصيص عملية تكون مسئولة عن هذا المورد، بحيث نمنع بقية العمليات من استخدام المورد مباشرة وإنما تتعامل هذه العمليات مع العملية المسئولة من المورد، وتقوم الأخيرة بالتعامل مع المورد.

مثلا مدير الطباعة في ويندوز يمثل عملية مسئولة عن الطابعة، حيث عندما يرسل أي برنامج (عملية) أمر طباعة، فإن الملف المراد طباعته سيرسل إلى مدير الطباعة ، ويتعامل مدير الطباعة مع الطابعة حيث يرسل لها ملف كلما فرغت من عملها حسب ورود الملفات إليه (القادم أولا يخدم أولا). مثلا الشكل (7-22) نجد أن هنالك ملفين أرسلا للطابعة ولكن قام باستلام هذه الملفات مدير الطباعة ثم ينظم هو التعامل مع الطابعة.



شكل رقم (7-22): مدير الطباعة في ويندوز.

7.6.4.2.2. نفي الاحتفاظ والانتظار

لا تبدأ العملية في العمل ما لم تتأكد من أن كل الموارد التي تحتاجها متوفرة وغير مستخدمة من قبل عمليات أخرى. ولكن هنا تظهر مشكلة هي أن معظم العمليات لا تستطيع ما تريد من موارد ما لم تبدأ العمل. أيضا هنالك مشكلة أخرى هي أنه حتى ولو توفرت كل الموارد للعملية وبدأت عملها فهذا يجعل الموارد محتكرة لأن العملية قد تحتاج مورد لعمل ما وبعد نصف ساعة ستحتاج المورد الثاني، فتكون العملية قد حجزت هذا المورد الأخير لمدة نصف ساعة دون أن تستفيد منه بقية الموارد.

7.6.4.2.3. نفي الاحتكار (عدم التخلي عن الموارد)

غير قابل للتطبيق: لان انتزاع مورد من عملية وهي تستخدمه أحيانا قد يتسبب في فشل العملية. مثلا إيقاف الطابعة وهي تطبع في ملف قد ينتج عنه طباعة غير مكتملة، وإيقاف مسجل الاسطوانات الضوئي (CD writer) وهو ينسخ بيانات في اسطوانة CD قد يتسبب في تلف الاسطوانة وفشل عملية النسخ.

7.6.4.2.4. نفي الانتظار الدائري

نرقم كل الموارد، ثم نسمح للعمليات بطلب الموارد بصورة تصاعدية ولا نسمح لعملية بأن تطلب مورد رقمه أصغر من المورد الذي طلبته من قبل، فمثلا إذا طلبت عملية ما المورد رقم 3، فلا نسمح لها بطلب مورد رقمه أقل من 3 ويمكنه طلب مورد رقمه أكبر من 3.

مثال

في الشكل (7-23)، إذا طلبت العملية أ الماسحة ورقمها هو 2، فلا يمكن لهذه العملية أن تطلب الطابعة لأن رقمها أقل من رقم الماسحة. افترض أن العملية ب قامت بطلب الشريط الممغنط (ذو الرقم 4)، فلا يمكن لهذه العملية طلب أي مورد رقمه أقل من 4، يعني لا يمكنها طلب الماسحة التي تستخدمها العملية أ. بهذه الطريقة لا يمكن أن يحدث انتظار دائري.

الطابعة	1

2	الماسحة
3	سواقة الأقراص
4	الشريط الممغنط
5	الراسم

شكل رقم (7-23): ترقيم الموارد تسلسليا.

7.7 محاكاة خوارزمية المصرف (*Banker's algorithms*)

لكتابة برنامج يطبق فكرة خوارزمية المصرف، مثلا لنفترض أن لدينا المحددات التالية:

3	1	1
3	2	1
2	3	5

Pmax

1	1	1
2	0	1
2	3	1

Pcurr

والموارد المتاحة : A=(3,1,1)

مثلا يمكننا إنشاء مصفوفة الموارد المطلوبة Pmax ومصفوفة الموارد الحالية Pcurr كما يلي:

```
Dim Pmax(2, 2), Pcurr(2, 2), avl(2) As Integer
Pmax(0, 0) = 3
Pmax(0, 1) = 3
Pmax(0, 2) = 2
Pmax(1, 0) = 1
Pmax(1, 1) = 2
Pmax(1, 2) = 3
Pmax(2, 0) = 1
```

```
Pmax(2, 1) = 1
Pmax(2, 2) = 5
Pcurr(0, 0) = 1
Pcurr(0, 1) = 2
Pcurr(0, 2) = 2
Pcurr(1, 0) = 1
Pcurr(1, 1) = 0
Pcurr(1, 2) = 3
Pcurr(2, 0) = 1
Pcurr(2, 1) = 1
Pcurr(2, 2) = 1
```

يمكن إنشاء مصفوفة للموارد المتاحة حاليا:

```
avl(0) = 3
avl(1) = 1
avl(2) = 1
```

معرفة احتياجات j العملية من الموارد :

```
For i = 0 To 2
    Console.Write(Pmax(j, i) - Pcurr(j, i) & "    ")
Next
```

ثم نختبر هل يكفي المتوفر من الموارد (avl) لعملية j كالتالي:

```
For i = 0 To 2
    If ((Pmax(j, i) - Pcurr(j, i)) <= avl(i)) Then
        Console.Write(avl(i) - (Pmax(j, i) - Pcurr(j, i)))
    Else
        Console.Write("not enough")
        ' stop
    End If
Next
```

7.8. تمارين محلولة

1. اذكر ثلاث أمثلة لموارد ؟ الطابعة ، القرص الصلب ، جدول قاعدة بيانات

2. تنقسم الموارد إلى نوعين ما هما ؟ قابلة للنزع وغير قابلة للنزع

3. اذكر مثال لمورد قابل للنزع ومثال لمورد غير قابل للنزع ؟ الطابعة (غير قابلة) ، الذاكرة (قابل للنزع)

4. هنالك أربع طرق للتعامل مع الاختناق ، اذكرها ؟

   1. تجاهله
   2. اكتشافه وعلاجه
   3. تجنبه
   4. منعه

5. ما الفرق بين التجنب والمنع ؟ التجنب هو حجز الموارد بصورة آمنة (safe) ، المنع هو نفي احد مسببات الاختناق

6. كيف يتم منع الاختناق ؟

   1. نفي المنع المتبادل
   2. نفي الاحتفاظ والانتظار
   3. نفي الاحتكار
   4. نفي الانتظار الدائري

7.ما هي مسببات الاختناق الأربعة ؟

   1. المنع المتبادل
   2. الاحتفاظ والانتظار

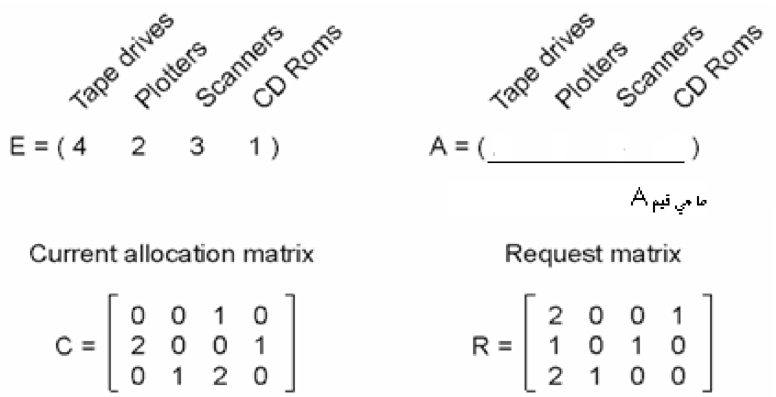
3. الاحتكار
4. الانتظار الدائري

8. ما هي خوارزمية النعامة ؟ ومتى نستخدمها ؟

تظاهر بأنه لا يوجد اختناق ، ونستخدمها إذا كان :

- الاختناق يحدث نادراً
- تكلفة الوقاية من الاختناق عالية جداً

9.مسألة



إذا كان E تمثل موارد النظام، والمصفوفة C تمثل الموارد المستخدمة منها، والمصفوفة R تمثل الموارد المطلوبة لإتمام العمل.

- اوجد الموارد المتوفر ة غير المستخدمة ؟
- وضح هل سيحدث اختناق أم لا ؟

الحل

(أ) الموارد المتوفرة A هي (0 0 1 2)

(ب) الحل

الخطوة الأولى

نقارن A مع صفوف C لنرى هل هنالك صف حجمه أقل أو يساوي A ؟

وبالمقارنة سنجد أن A يكفي للصف الثالث (العملية الثالثة) حيث يتساويان

وبإعطاء الموارد الموجودة في A للعملية الثالثة ستصبح A فارغة (0 0 0 0)

وستكمل العملية الثالثة عملها فتحرر الموارد التي كانت تحتجزها (C) والموارد التي طلبتها (R) فتكون قيمة A الجديدة هي (0 2 2 2)

وستصبح C و R كالتالي:



الخطوة الثانية

الآن سنقارن new A مع صفوف R فنجد أنها تكفي لإكمال الصف الثالثة

بعد الانتهاء من الصف الثالثة ستكون الموارد المحررة هي (1 2 2 4).

الآن ستصبح C و R كالتالي:



الخطوة الثالثة

سيتوفر لدينا الآن من الموارد التالي ()A كافية لتنفيذ الصف المتبقي (الأول) وهذا يعني أنه لا يوجد اختناق (حالة آمنة).

10.مسألة

	Tape drives	Plotters	Scanners	CD Roms
E =	3	1	2	2

	Tape drives	Plotters	Scanners	CD Roms
A =				

Current allocation matrix

$$C = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

ضح هل يوجد اختناق أم لا ؟

<u>الحل</u>

الموارد المتوفرة هي

$$A = (1 \quad 0 \quad 0 \quad 1)$$

ننفذ العملية الأولى (الصف الأول في R) وبعد تنفيذها ستحرر الموارد التالية:

<u>(1 0 1 1)</u>

وستصبح C و R كالتالي:

Current allocation matrix

$$C = \begin{bmatrix} \cancel{0} & \cancel{0} & \cancel{1} & \cancel{0} \\ 2 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} \cancel{1} & \cancel{0} & \cancel{0} & \cancel{1} \\ 1 & 0 & 1 & 0 \\ 2 & 1 & 0 & 0 \end{bmatrix}$$

ننفذ العملية الثانية وبعد تنفيذها ستحرر الموارد التالية (2 1 0 3).

الآن ستصبح C و R كالتالي:

Current allocation matrix

$$C = \begin{bmatrix} \cancel{0 \quad 0 \quad 1 \quad 0} \\ \cancel{2 \quad 0 \quad 0 \quad 1} \\ 0 \quad 1 \quad 1 \quad 0 \end{bmatrix}$$

Request matrix

$$R = \begin{bmatrix} \cancel{1 \quad 0 \quad 0 \quad 1} \\ \cancel{1 \quad 0 \quad 1 \quad 0} \\ 2 \quad 1 \quad 0 \quad 0 \end{bmatrix}$$

نجد أن المتبقي(2 1 0 3) لا يكفي للعملية الثالثة وهذا يعني أن هنالك اختناق.

11. كيف تتم معالج الاختناق ؟

1. انتزاع المورد من أحدي العمليات المشاركة في الاختناق

2. إيقاف (قتل Kill) بعض العمليات التي تشارك في الاختناق وبالتالي استخدام مواردها لبقية العمليات

12. مسألة

إذا كان لدى ثلاث عمليات A,B,C لديها الموارد المبينة بالعمود Has وتحتاج أن تصل مواردها للعمود Max لتنفذ عملها ، وليدنا ثلاث موارد متاحة (كما مبين في الجدول أعلاه).

	Has	Max
A	3	9
B	2	4
C	2	7

Free: 3

وضح كيف يمكننا تنفيذ البرامج الثلاث دون حدوث اختناق (حالة آمنة) ؟

وضح كيف يمكن أن يحدث اختناق إذا وزعنا الموارد بصورة أخرى ؟

<u>الحل</u>

(أ)

	Has	Max
A	3	9
B	4	4
C	2	7

Free: 1

	Has	Max
A	3	9
B	0	–
C	2	7

Free: 5

	Has	Max
A	3	9
B	0	–
C	7	7

Free: 0

	Has	Max
A	3	9
B	0	–
C	0	–

Free: 7

(ب)

	Has	Max
A	4	9
B	2	4
C	2	7

Free: 2

	Has	Max
A	4	9
B	4	4
C	2	7

Free: 0

	Has	Max
A	4	9
B	—	—
C	2	7

Free: 4

7.9. تمارين غير محلولة

1. إذا كان لدي نظام يحتوي على العمليات التالية مع ما تملك (المستخدم) وما يكفيها من موارد (أي أن ما تحتاج من موارد هو مايكفي ناقص ما تستخدم)، كما في الجدول التالي:

	المستخدم				ما يكفي				المتاح			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	1	0	1	2	2	2	1	2	2	6	2	1
P1	1	2	0	0	1	7	5	0				
P2	1	3	4	3	2	3	5	6				
P3	0	6	4	2	0	6	5	2				
P4	0	2	3	3	0	6	5	6				

- مثلا العملية P0 تستخدم 1 من المورد A، و تحتاج 2 من هذا المورد، إذن تحتاج 1 من المورد A. أيضا تستخدم 2 من المورد D ويكفيها 2، أي لاتحتاج أي مورد إضافي من D.

أجب على الآتي:

- كون محددة المطلوب (ما يكفي – المستخدم) ؟
- هل النظام آمن (safe state) ؟ أي هل سيحدث إختناق أم لا ؟
- إذا كانت طلبات العملية P0 هي 0 1 1 0 هل سيتم خدمتها مباشرة ؟
- إذا كانت طلبات العملية P1 هي 0 3 3 0 هل سيتم خدمتها مباشرة ؟

2. معطى محددة الموارد (على اليسار) ومحددة الاحتياجات (على اليمين)، هل سيحدث إختناق أم لا (خوارزمية المصرف متعددة الموارد) ؟

Process	Tape drives	Plotters	Printers	CD ROMs
A	3	0	1	1
B	0	1	0	0
C	1	1	1	0
D	1	1	0	1
E	0	0	0	0

Resources assigned

Process	Tape drives	Plotters	Printers	CD ROMs
A	1	1	0	0
B	0	1	1	2
C	3	1	0	0
D	0	0	1	0
E	2	1	1	0

Resources still needed

E = (6342)
P = (5322)
A = (1020)

الباب الثامن:
إدارة الذاكرة الرئيسية

الباب الثامن

إدارة الذاكرة الرئيسية

Main Memory Management

لا يعمل جهاز الحاسب بدون ذاكرة رئيسية (*ram*)، وعندما تذهب لتشتري جهاز حاسب فتجد مواصفات مثل *2MB Ram*، والتي تعني أنك حاسبك يمتلك ذاكرة رئيسية حجمها *2* ميغابايت أي *1024 * 1024 *2 = 2097152* بايت، والبايت يعادل حرف،أي أكثر من أثنين مليون حرف.

تعتبر الذاكرة الرئيسية متطلباً أساسياً لتنفيذ البرامج، فلا يمكن تنفيذ برنامج ما لم يحضر إليها. لأن المعالج لا يتعامل إلا مع الذاكرة الرئيسية والمسجلات التي داخله. يكون تعامل المعالج مع المسجلات سريعا جدا (دورة ساعة واحدة (*one CPU clock*) أو أقل)، بينما الوصول للذاكرة الرئيسية يكون أقل سرعة (عدة دورات). تعتبر الذاكرة المخبأة (الكاش) بين الذاكرة الرئيسية والمسجلات.

كبر حجم الذاكرة يمكنها من خدمة عدد كبير من العمليات، بينما سرعتها تزيد من سرعة الوصول للمعلومة فيها. لذلك لابد من إدارتها بالطريقة المثلى التي تؤثر تأثير إيجابيا على أداء الحاسب.

الكل يريد حاسب بذاكرة رئيسية كبيرة وسريعة وغير متطايرة وفوق ذلك رخيصة. لكن هذه المواصفات لا تجتمع في ذاكرة واحدة (حتى كتابة هذه السطور). وتوجد هذه الصفات متفرقة ين عدة ذواكر، فمثلا صفة السرعة توجد في المسجلات والكاش (لكنها صغيرة وغالية)، بينما القرص الصلب كبير ورخيص لكنه بطئ جدا.

للإستفادة من هذه الذواكر المختلفة يمكن استخدام القرص الصلب مثلا للتخزين الدائم والكبير، بينما نستخدم الذاكرة الرئيسية والكاش والمسجلات للتنفيذ السريع. يدير هذه الذواكر جزء من نظام التشغيل يسمى مدير الذاكرة ويتمثل عمله في الآتي:

- تتبع أجزاء الذاكرة المستخدمة والغير مستخدمة.
- حجز الذاكرة للعمليات التي تحتاجها.

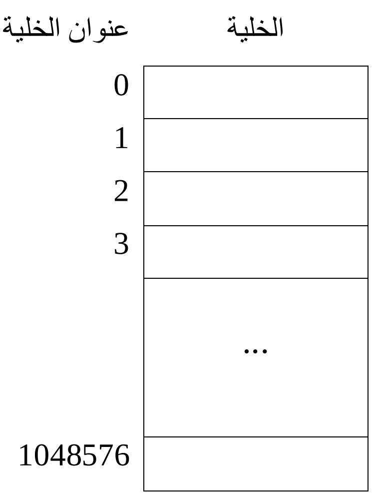
- تحديد أين يتم وضعها.
- تحميل العمليات بالذاكرة.
- تفريغ (تحرير) الذاكرة عند إنتهاء العمليات من استخدامها.
- إدارة التبديل بين الذاكرة الرئيسية والقرص الصلب (*swap*).
- حماية العمليات في الذاكرة من بعضها البعض ومن نظام التشغيل.

8.1. بنية الذاكرة الرئيسية

تتكون الذاكرة من مجموعة من الخلايا الثنائية موزعة بطريقة تشبه المصفوفة حيث يحدد عدد الخلايا في السطر الأول طول الكلمة، وتحمل كل خلية عنوان (رقم) فريد لا يتكرر تستطيع من خلاله وحدة المعالجة تحديد مكان الكلمة المطلوبة في الذاكرة. يقاس حجم الذاكرة عادة بمجموع الخلايا الثنائية المتوفرة. مثلا إذا كان في حاسبك ذاكرة حجمها 1 ميغابايت (*MB*)، فهذا يعني أن لدينا:

$$1 * 1024 * 1024 = 1048576 \text{ بايت}$$

البايت يخزن رمز أو حرف، فإذا كان كل بايت يمثل خانة بالذاكرة فسيكون لدينا أكثر من مليون خانة بالذاكرة وكل خانة لديها عنوان يختلف عن عنوان الخانة الأخرى وبالتالي سيكون لدينا أكثر مليون عنوان، الشكل (8-1).



شكل رقم (8-1): عناوين ذاكرة حجمها واحد ميغابايت.

8.2. أهداف مدير الذاكرة

هنالك العديد من الأهداف من وراء تصميم مدير الذاكرة، مثل:

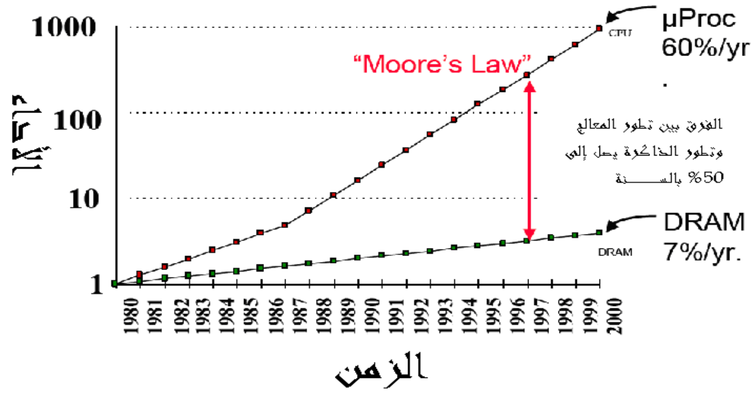
- حجز المواقع وتحريرها.
- التعامل مع هرمية الذاكرة.
- العناوين المنطقية: استخدام العناوين المنطقية للتعامل مع الذاكرة.
- الحماية: حماية البرامج عن بعضها البعض.
- المشاركة: توفير المشاركة بين البرامج دون التأثير على الحماية.
- توسيع الذاكرة: تمديد الذاكرة لتستوعب برامج أكبر من حجمها وذلك باستخدام جزء من القرص الصلب.

8.2.1. حجز المواقع

إذا أردت تشغيل برنامج فإن مدير الذاكرة سيحجز مساحة بالذاكرة لهذا البرنامج ثم إذا ما أنتهينا من استخدام البرنامج وقمنا بطلب إغلاقه، سيقوم مدير الذاكرة بتحرير ما حجز لهذا البرنامج من الذاكرة وتفريغ مكانه للاستفادة منه في تحميل برامج أخرى.

8.2.2. هرمية الذاكرة

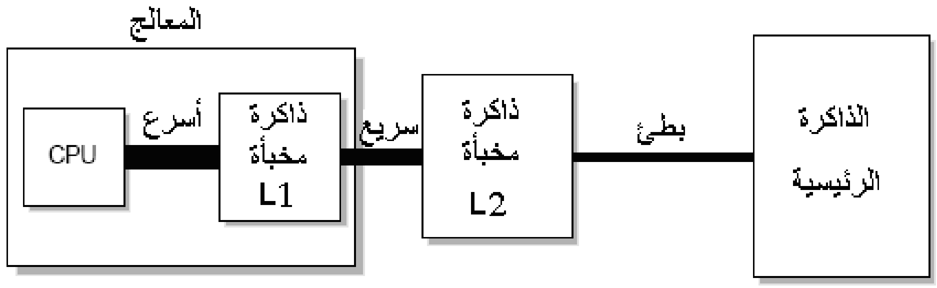
في الثمانينات كانت الذاكرة تعمل بسرعة المعالج، ثم بدأ سباق التطور، فوصل المعالج بسرعته إلى أضعاف ما وصلت إليه الذاكرة، الشكل (8-1). وأصبحت سرعة الذاكرة الرئيسية لا تستطيع مجاراة سرعة المعالج، وهذا تسبب في أبطاء المعالج لسرعته حتى يتمكن من التعامل مع الذاكرة.



شكل رقم (8-1): الفرق بين سرعة المعالج والذاكرة الرئيسية.

لم تكن هنالك ذاكرة مخبأة في 1980، لماذا ؟

تم علاج هذا الفارق بين سرعتي المعالج والذاكرة الرئيسية بوضع ذاكرة سريعة بينهما هي الذاكرة المخبأة، الشكل (8-2)، والتي تعمل بسرعة المعالج أو قريبة منه.



شكل رقم (8-2): الذاكرة المخبأة بين الذاكرة الرئيسية والمعالج.

عند تشغيل برنامج كبير (أكبر من حجم الذاكر الرئيسية) سيكون بالقرص، ثم سيحمل مدير الذاكرة جزء من هذا البرنامج إلى الذاكرة الرئيسية (نسخة من الجزء وستظل نسخة أخرى بالقرص ، أي أن البرنامج كاملا سيكون بالقرص). هذا الجزء هو الذي نحتاجه الآن، سيحمل جزء من الجزء الذي بالذاكرة الرئيسية إلى الذاكرة المخبأة، بالتالي سيكون هذا الجزء الصغير مكرر في ثلاث أماكن ، في القرص وبالذاكرة الرئيسية وفي الذاكرة المخبأة. ثم سنأخذ كل مرة أمر وبضع بيانات من الذاكرة المخبأة لتنسخ في مسجلات المعالج حيث يتم تنفيذها. سنجد أن النسخ الآن أصبحت أربعة نسخ،

وقد يكون لدينا خمس نسخ إذا استخدمنا مستويين من الذاكرة المخبأة (L1 و L2)، الشكل (8-2). هذا سيجعل مهمة مدير الذاكرة أصعب، فعليه التأكد من أن هذه النسخ الموزعة على عدة ذواكر متوافقة (consistent). هذا بالإضافة إلى متابعة حركة البيانات بين هذه الذواكر المختلفة المستويات.

8.3. مواصفات الذاكرة المثالية

يتمنى كل شخص منا سواء كان مستخدما للحاسب أم مبرمجا أن تكون لديه ذاكرة بالمواصفات التالية:

- كبيرة
- سريعة
- رخيصة.
- غير متطايرة (non volatile) – أي لا تفقد محتواها.

ولكن هيهات هيهات، فلم تجتمع هذه المواصفات في نوع واحد من الذواكر حتى يومنا هذا. يمكن استخدام عدة ذواكر مختلفة تكمل بعضها البعض لتكون هرمية مثالية. فمثلا نستخدم الذاكرة المخبأة (الكاش) لنحصل على السرعة العالية، ولكنها في جانب الآخر، صغيرة و متطايرة (لا تحتفظ بمحتوياتها)، وغالية .

لذلك يمكنني استخدام الذاكرة الرام معها لأتمكن من وضع برامج كبيرة، فالرام تعتبر سريعة نوعا ما (متوسطة السرعة) ، ويمكنني استخدام حجم أكبر منها لأن سعرها معقول (متوسطة السعر)، لكن كسابقتها في عدم مقدرتها لحفظ البيانات بصورة دائمة. لذلك لن استطيع الاستغناء عن الذاكرة الثانوية التي تتميز عن بقية الذوالكر السالف ذكرها في أنها تحتفظ بمحتوياتها حفظا دائما. هذا النوع من الذواكر. بالإضافة إلى كونه غير متطاير (لا يفقد محتواه)، نجد أنه رخيص.

8.4. مثال توضيحي

أراد أحمد وهو مدخل بيانات طباعة كتاب باستخدام ميكروسوفت وورد، فقام:

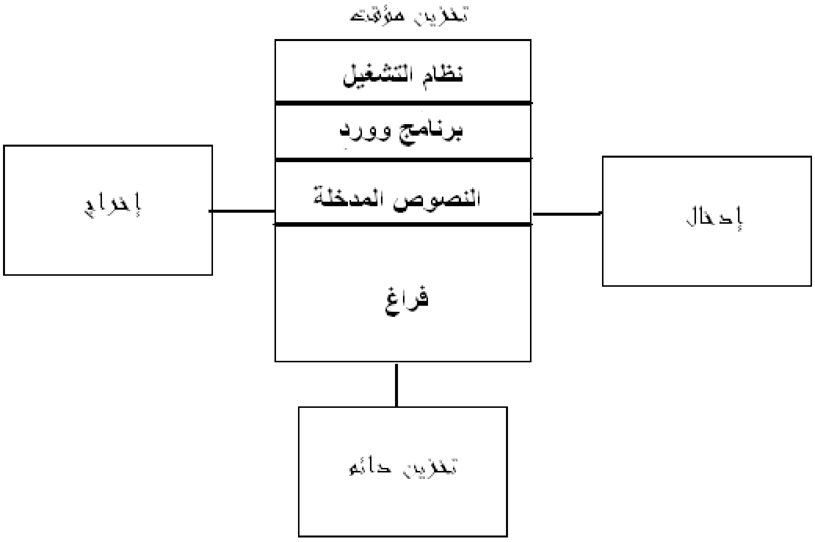
- بفتح الحاسب فظهر سطح المكتب.
- نقر نقرا مزدوجا على الوورد ففتح البرنامج وظهرت نافذته أمام أحمد.
- بدأ أحمد بإدخال البيانات.
- أدخل جزء من البيانات مثلا 3 صفحات ثم قام بحفظ الملف (ملف – حفظ).
- أدخل 4 صفحات أخرى ولكن قبل حفظ الملف مرة أخرى إنفصل التيار الكهربي عن الحاسب فأغلق الجهاز.
- إذا فتح أحمد الحاسب مرة أخرى وفتح ملف وورد السابق حفظه ؟ هل سيجد 3 صفحات أم 7 صفحات ؟ ولماذا ؟

داخليا يجري الآتي:

عند فتح الحاسب يتحمل نظام التشغيل في جزء من الرام، فظهور سطح المكتب أمام أحمد دلالة على ذلك. النقر المزدوج على أيقونة وورد هو طلب من أحمد لنظام التشغيل ليقوم بإنشاء عملية جديدة، وظهور نافذة وورد أمامه دليل على أن العملية تم إنشاءها وهي تعمل، أن برنامج وورد الآن بالذاكرة (في مساحة أخرى غير المساحة التي تحمل بها نظام التشغيل).

أين كان برنامج وورد قبل أن يعمل ؟ عندما بدأ أحمد بالإدخال بدأت تظهر المدخلات على الشاشة وهذا يدل على أن البيانات تنتقل من لوحة المفاتيح إلى الذاكرة ومن الذاكرة إلى الشاشة دون أن يكون لها علاقة بالقرص الصلب أو أي ذاكرة ثانوية أخرى.

عملية حفظ الملف تعني نسخ ما أدخله أحمد (3 صفحات) والذي يوجد الآن بالرام إلى القرص الصلب.



شكل رقم (3-8): شكل البرامج البيانات في الذاكرة الرئيسية.

أدخل أحمد 4 صفحات أخرى ولكن قبل أن يقوم بالحفظ أغلق الجهاز، ستفقد الرام محتواها لانها متطايرة (وبالتالي سيفقد أحمد الأربعة صفحات الأخيرة لأنه لم يحفظها بالذاكرة الثانوية).

عندما يفتح أحمد الجهاز مرة ثانية ويفتح ملفه السابق سيجد فقط 3 صفحات، أين بقية الصفحات التي أدخلها ؟

من المثال أعلاه تلاحظ أنه لا غنى لنا عن الذاكرة الثانوية في الحفظ الدائم للمعلومات. فعندما أغلق الجهاز فقدنا كل محتويات الرام، فإضطر أحمد عند فتحه للجهاز في المرة الثانية لتشغيل وورد مرة أخرى لأنه أصبح غير موجود بالرام، وأعاد إدخال الصفحات الأربعة (التي فقدت) مرة ثانية. كذلك لاغنى لنا عن الذاكرة الرئيسية في تشغيل البرامج، ونحتاج الكاش في تسريع العمل.

كلما نفتح الحاسب يقوم برنامج موجود بالروم (rom) بتحميل نظام التشغيل بالرام، أي أن الروم هي المسئولة عن تحميل نظام التشغيل، أما بقية البرامج فنحن من يطلبها ونظام التشغيل هو من يقوم بتشغيلها لنا. إذن الروم تشغل نظام

التشغيل ونظام التشغيل يشغل برامجنا.

8.5. أنواع تعدد المهام

من الشكل (8-3)، نجد أن نظام التشغيل وبرنامج وورد والبيانات المدخلة كلها موجودة بالذاكرة الرئيسية (الرام)، لم وضعت بهذه الطريقة ؟ أين سيتم وضع البرامج الأخرى إذا قمت بطلب تشغيلها، ومن الذي يتولى تحميل البرامج بالذاكرة وكيف نتأكد أن برنامج لم يتحمل في مكان برنامج آخر؟ كل هذه الاستفسارات والأسئلة تدور حول الذاكرة وكيف يتم تخزين البرامج والبيانات فيها.

تعتمد الإجابة على أسئلتنا هذه على مدير الذاكرة وطريقة عمله. سنقوم فيما يلي بشرح إدارات الذاكرة المختلفة لتخزين عدة برامج (تعدد المهام) بالذاكرة.

سنبدأ بتوضيح طريقة إدارة الذاكرة في النظم القديمة (التي لم تعد مستخدمة حاليا) ولكنها ستساعدنا في فهم الحديث، ثم نوضح بعدها كيف يدير نظام التشغيل الحديث الذاكرة.

حيث سنتحدث عن:

- الذاكرة أحادية المهام (النظم القديمة).
- الذاكرة متعددة المهام (النظم الحديثة).
- التجزئة الثابتة (fixed partitions).
- التجزئة الديناميكية.
- تجميع الفراغات.
- الذاكرة بالصفحات.
- الذاكرة بالتقطيع.

8.6. نظام التشغيل أحادي المهام

قديما كانت الذاكرة صغيرة وتعمل بالنظام الأحادي (single program)، حيث يسمح للمستخدم بتشغيل برنامج واحد فقط بالإضافة إلى نظام التشغيل وبالتالي فإن الذاكرة الرئيسية تكون مقسومة إلى جزئين، جزء مخصص لنظام التشغيل وجزء مخصص للمستخدم لينفذ فيه برنامج واحد فقط كل مرة، هذا النوع من نظم التشغيل يسمى نظم التشغيل أحادية المهام (single task) أو أحادية البرامج (single program)، شكل رقم (8-4).



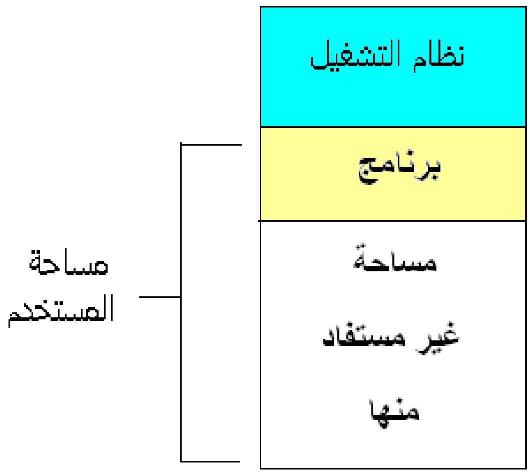
شكل رقم (8-4): شكل الذاكرة في النظام أحادي المهام.

8.6.1. عيوب النظام الأحادي

هنالك برنامج واحد فقط بالذاكرة حتى لو كان هذا البرنامج صغير جدا ولا يستغل إلا القليل من مساحة المستخدم، وهذا يتسبب في الآتي:

- إهدار مساحة الذاكرة، وذلك للآتي:
  - هنالك مساحات فارغة تكفي لتحميل برامج أخرى ولكن النظام لا يسمح بتحميل أكثر من برنامج، شكل رقم (8-5).
  - قد يكون بالبرنامج المنفذ بيانات أو أوامر نادرة الاستخدام، فهي تحجز حيزا بالذاكرة (لأن البرنامج لابد من أن يكون كاملا بالذاكرة).
  - لن نستطيع تشغيل برامج حجمها أكبر من مساحة المستخدم.

- إهدار زمن المعالج : غير مستفاد من زمن المعالج، لأن المعالج ينفذ برنامج واحد فقط في اللحظة الواحدة، وإذا توقف هذا البرنامج عن العمل لأي سبب كأن يكون في انتظار دخل أو حدوث حدث، في هذه الحالة سيكون المعالج عاطلا لا يعمل شيئا وغير مستفاد منه.



شكل رقم (8-5): إهدار مساحة المستخدم.

8.6.2 كيف يدير نظام التشغيل الذاكرة

يقوم مدير الذاكرة بمعرفة حجم البرنامج الذي طلبنا تنفيذه ، ثم يقارن هذا الحجم مع حجم مساحة المستخدم، إذا كان حجم البرنامج أصغر أو يساوي هذه مساحة المستخدم سيتم تحميله في الذاكرة، وإلا فسيمنع تحميله وقد تظهر رسالة توضح ذلك (لا توجد ذاكرة كافية *not enough memory*).

أيضا يقوم مدير الذاكرة بحماية منطقة نظام التشغيل من برامج المستخدم بحيث يتم وضع العنوان الفاصل بين نظام التشغيل ومساحة المستخدم في مسجل يسمى مسجل الحماية (*Protection register*)، عندما ينفذ برنامج المستخدم أي عملية وصول للذاكرة سيقارن نظام التشغيل العنوان المستخدم في الأمر مع العنوان المخزن في مسجل الحماية إذا كان أعلى منه، يمنع البرنامج من تنفيذ الأمر لأنه تعدي للحدود الإقليمية وتجاوز لمنطقة المستخدم ومحاولة للوصول إلى منطقة نظام التشغيل.

من السهل على مدير الذاكرة إدارة ذاكرة مثل هذه الذاكرة الأحادية، فكل ما عليه هو حماية نظام التشغيل واختبار حجم البرنامج هل تكفي الذاكرة لتحميله أم لا ؟ وهذه تعتبر الحسنة الوحيدة في هذا النظام. مثال لهذا النوع من النظم هو نظام التشغيل DOS.

8.7. نظام التشغيل متعدد المهام

إذا سمح نظام التشغيل للمستخدم بتشغيل أكثر من برنامج في وقت واحد، فأنت تتعامل مع نظام تشغيل متعدد المهام (*multitasking*) أو متعدد البرامج (*multiprogramming*). حيث يمكن تحميل أكثر من برنامج بالذاكرة. مثلا في نظام التشغيل ويندوز يمكنك تشغيل برنامج وورد لتكتب وثيقة والإستماع إلى مشغل الوسائط (*media player*) وإنزال ملفات من الإنترنتت في نفس الوقت. وهذا يدل على أن ويندوز نظام متعدد المهام.

إذا أردت أن تقوم بمثل هذا العمل في نظام تشغيل أحادي المهام (مثل DOS) فلن تستطيع، وإنما عليك تشغيل هذه البرامج في أوقات مختلفة (واحد تلو الآخر.).

السؤال الهام الذي يطرح نفسه هنا هو : هل هنالك تعدد مهام حقيقي ؟ الإجابة ترتبط بنوع العتاد الذي نستخدم.

إذا كنت تستخدم حاسب ذو معالج واحد (*single processor*)، فلن يكون هنالك تعدد مهام حقيقي وإنما استغلال جيد لزمن المعالج. أما إذا كنت تستخدم معالج بقلبين (*dual core*) أو كنت تستخدم حاسب متعدد المعالجات (*multiprocessor*) فسيكون نظامك يعمل بتعدد مهام حقيقي.

8.7.1. مثال تشبيهي

لتوضيح الفرق بين تعدد المهام الحقيقي والوهمي دعنا نشرح ذلك بمثال تشبيهي.

8.7.1.1. تعدد المهام الوهمي (الموظف النشيط)

بالرجوع للمثال التشبيهي بالباب الثالث (3.8)، نجد أنه إذا كان الموظف نشيط وسريع فيمكنه العمل بطريقة متعددة (غير حقيقية)، حيث سينجز أعماله كما يلي:

- يدخل عميل ويبدأ في خدمته.
- إذا أحتاج الموظف بعض البيانات من العميل سيطلب منه تعبئة هذه البيانات.
- في هذه اللحظة يستدعي الموظف عميل آخر ويبدأ بخدمته ريثما ينتهي العميل السابق من تعبئة بياناته (الاستفادة من وقت الفراغ).
- إذا أحتاج العميل الثاني مثلا لتصوير مستندات، سيذهب لفعل ذلك.
- العميل الأول يعبئ في بيانات والثاني ذهب ليصور مستندات، الآن الموظف لا يعمل (فارغ).
- يمكن للموظف أن يحضر عميل ثالث ويبدأ في إنجاز معاملته.
- إذا فرغ العميل الأول من تعبئة البيانات سيكون جاهزا لإكمال معاملته.
- بعد فراغ الموظف من العميل الثالث سيدخل العميل الأول ليكمل له معاملته، أو قد يدخل عميل رابع، حسب ما يرى (الجدولة)، وهكذا.

بهذه الطريقة نقول أن الموظف يخدم عدد من العملاء في وقت واحد، لكن الحقيقة أنه لا يستطيع التعامل ولا الاستماع ولا التكلم إلا من عميل واحد في اللحظة الواحدة، و لكن استغلاله الجيد لوقت فراغه مكنه من خدمة عملاء كثر (كأنه يخدم عدة عملاء في وقت واحد). هذه الطريقة تشبه عمل المعالج (الموظف) مع البرامج (العملاء) في مفهوم تعدد المهام الوهمي.

8.7.1.2. أحادية المهام

أما بالنسبة لنظام التشغيل أحادي المهام فهو يشبه الموظف الكسلان، حيث سيبدأ الموظف بخدمة عميل ثم إذا قام العميل بتعبئة نموذج أو تصوير مستند أو البحث عن معلومة في أوراقه، ظل الموظف جالسا لا يؤدي أي مهام حتى يفرغ العميل

من تعبئيه بياناته ثم يرجع مرة أخرى للموظف وقد يتوقف العميل أكثر من مرة لتكملة أوراق أو أي شي آخر ويظل الموظف منتظرا هذا العميل.

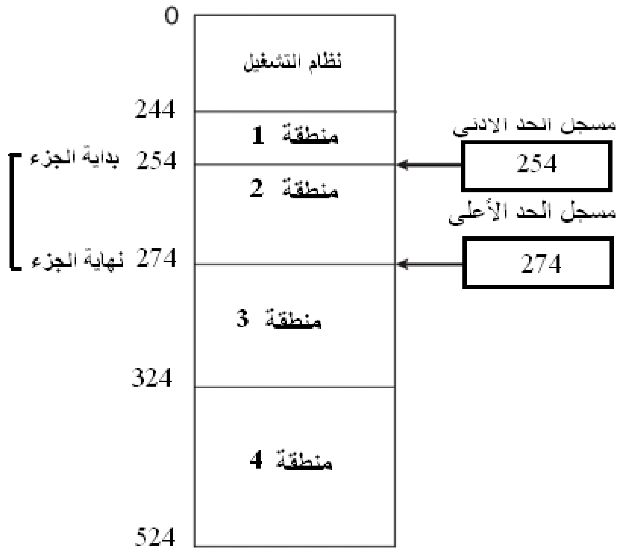
8.7.1.3. تعدد المهام الحقيقي (عدد من الموظفين)

إذا كان لدينا عدد ثلاث موظفين مثلا، فيمكن لكل موظف أن يخدم عميل ، بالتالي سيكون عدد العملاء الذين سيتم خدمتهم في اللحظة الواحدة يساوي عدد الموظفين الموجودين. هنا يعتبر التعدد تعدد حقيقي للمهام، لأنه من الممكن للموظفين الثلاث التحدث والاستماع والتعامل مع ثلاث عملاء في وقت واحد.

إذا كان لديك معالج واحد فليس هنالك تعدد مهام حقيقي وإنما استغلال جيد لزمن المعالج. أما تعدد المهام الحقيقي فيتوفر في الحواسيب التي تمتلك أكثر من قلب بالمعالج (multi-core)، أو أكثر من معالج في الحاسب (multiprocessor).

8.8. التجزئة الثابتة

يقسم نظام التشغيل الذاكرة إلى مناطق ثابتة مختلفة الأحجام وذلك لإتاحة الفرصة لتحميل برامج بأحجام مختلفة في هذه المناطق. تستخدم إدارة الذاكرة مسجلي حدود لكل منطقة من مناطق الذاكرة حيث يخزن في المسجل الأول الحد الأعلى للمنطقة بينما يخزن في المسجل الثاني الحد الأسفل للمنطقة.



شكل ر قم (8-6): شكل الذاكرة لجدول الحجز (8-1).

8.8.1. كيف يعمل مدير الذاكرة

يقوم مدير الذاكرة باستخدام جدول يسمى جدول الحجز، يحتوي هذا الجدول على رقم كل منطقة وحجمها وأين توجد بالذاكرة وهل هي مستخدمة أم فارغة.

رقم المنطقة	حجم المنطقة	بدايتها	استخدامها
1	10	244	فارغ
2	20		فارغ
3	50		مشغول
4	200		فارغ

جدول رقم (8-1): جدول حجز الذاكرة.

مثلا جدول الحجز (8-1) يمثل الذاكرة بالشكل (8-6). حيث نستنتج منه الآتي:

- درجة تعدد المهام هي 4 (يمكن تحميل 4 برامج في وقت واحد).
- المنطقة 3 مشغولة (بها برنامج الآن، ولا يمكن تحميل برامج بها).
- حجم أكبر برنامج يمكن تحميله هنا هو 200 كيلوبايت (منطقة رقم 4).

- شكل الذاكرة للجدول (8-1) هي الشكل (8-7).



شكل ر قم (8-7): شكل الذاكرة لجدول الحجز (8-1).

8.8.2. خوارزميات التسكين

إذا كان هنالك عدة مناطق فارغة ونريد تحميل برامج بها، فكيف سيتم تحميل هذه البرامج ؟ هنالك خوارزميات مختلفة لتحميل البرامج بالذاكرة مثل:

- الأنسب (best-fit): يتم وضع البرنامج في أقل فراغ يمكن أن يستوعبه، حيث سيتم البحث عن كل الفراغات المتاحة بالذاكرة وإختيار أقل فراغ يمكن أن يستوعب البرنامج (العملية) التي نريد تحميلها بالذاكرة.

- الأول (first-fit): يتم وضع البرنامج في أول فراغ يمكن أن يستوعبه، هذه الخوارزمية لا تحتاج بحث عن كل الفراغات الموجودة بالذاكرة، وإنما ستضع البرنامج المراد تحميله في أول فراغ تجده بالذاكرة يكون حجمه أكبر أو يساوي حجم البرنامج.

- الأكبر (worst-fit): يتم وضع البرنامج في أكبر فراغ يمكن أن يستوعب البرنامج، حيث سيتم البحث عن كل الفراغات الموجودة بالذاكرة وإختيار

أكبرها حجما لوضع البرنامج المراد تحميله فيه (بحيث يكون الفراغ أكبر أو يساوي حجم البرنامج).

مثال (8-1)

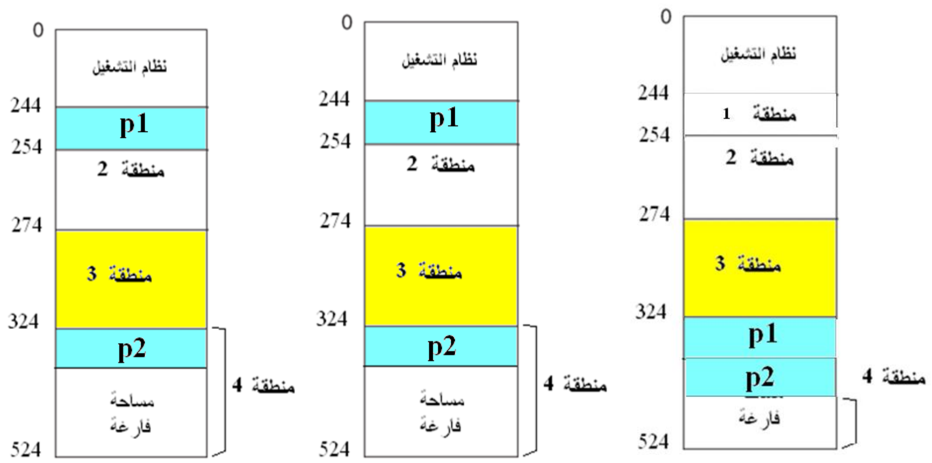
إذا كان لدينا ثلاث برامج بالأحجام P1=10، P2=50، P3=200، معطى جدول الحجز (8-1)، فكيف سيتم تحميل هذه البرامج باستخدام خوارزميات التسكين أعلاه ؟

الحل

طريقة الأنسب (best-fit): سنضع P1 في المنطقة 1، و P2 في المنطقة 4، و P3 سيتنتظر (يمكن تحميله لأنه لا يوجد فراغ يكفيه)، أنظر الشكل (8-8).

طريقة الأول (first-best): سنضع P1 في المنطقة 1، و P2 في المنطقة 4، و P3 سيتنتظر (يمكن تحميله لأنه لا يوجد فراغ يكفيه)، أنظر الشكل (8-8).

طريقة الأسوأ (worst-fit): سنضع P1 في المنطقة 4، و P2 في المتبقي من المنطقة 4 (200-10=190)، P3 لايمكن تحميله بالذاكرة لأنه لا يوجد فراغ يكفي.



شكل رقم (8-8): شكل الذاكرة للخوارزميات الثلاثة عد تحميل البرامج في المثال (8-1).

مثال (8-2) – غير محلول

إذا كان لدنا جدول الحجز التالي:

رقم المنطقة	حجم المنطقة	بدايتها	استخدامها
1	10	244	فارغ
2	15		فارغ
3	40		فارغ
4	200		فارغ
5	100		فارغ
6	5		فارغ
7	400		فارغ

1. أرسم شكل الذاكرة لجدول الحجز أعلاه.
2. وضح كيف يمكن تحميل العمليات التالية في الذاكرة :

باستخدام P1=5, p2=100, p3=40, p4=500, p5=200 :

- الأول ff
- الأنسب bf
- الأسوأ wf

مثال (3-8)

إذا كانت لدينا ذاكرة تتكون من الأقسام التالية (بالترتيب):

100k, 500k, 200k, 300k, 600k

وضح كيف تضع كل من الخوارزميات، الأول (ff)، الأنسب (bf)، والأسوأ (wf)، العمليات التالية بالذاكرة:

P1=212k, p2=417k, p3=112k, p4=426k

أي خوارزمية هي الأفضل في استخدام الذاكرة ؟

تنبيه: إذا تم تحميل عملية في قسم، يمكن إنشاء قسم جديد من الفراغ المتبقي بهذا القسم (إذا كان يكفي لتحميل عملية أخرى).

الحل

(أ) الأول (ff):

ضع 212 في القسم 500، 417 في القسم 600، 112 في القسم 288 (قسم نتج من 288=212-500)، 426 تنتظر.

(ب) الأفضل (bf):

نضع 212 في 300، 417 في 500، 112 في 200، 426 في 600

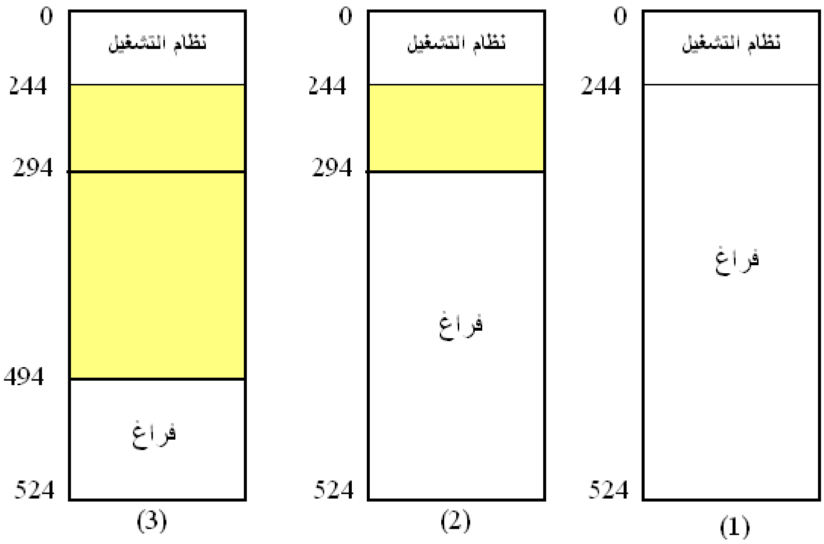
(ت) الأسوأ (wf):

212 في 600، 417 في 500، 112 في 388، 426 تنتظر.

في هذا المثال نجد أن الأفضل (bf) هي أفضل خوارزمية.

8.9. التجزئة الديناميكية

تكون مساحة المستخدم بالذاكرة في البداية غير مقسمة، ثم ينشأ جزء كلما تحمل برنامج بالذاكرة (يكون بحجم البرنامج) وتكون باقي المساحة فارغة، ثم إذا تم تحميل برنامج آخر،سيبنى جزء ثاني بحجم البرنامج الثاني، وهكذا، الشكل (8-9).



شكل رقم (8-9): (1) ذاكرة المستخدم فارغة في البداية (2) الذاكرة بعد تحميل برنامج بحجم 50 كيلوبايت (3) الذاكرة بعد تحميل برنامج آخر بحجم 200 كيلوبايت.

8.9.1. الفراغات (fragmentations)

الفراغات الخارجية External Fragmentation مساحة فارغة غير متلاصقة تنتج بسبب تحميل العمليات وإخراجها من الذاكرة، أنظر الشكل (8-9)، و الشكل (8-10).

الفراغات الداخلية Internal Fragmentation: قد يتم حجز منطقة لعملية بحيث تكون العملية اصغر من المنطقة مما يولد فراغ داخلي لا يمكن استخدامه ، مثل الفراغ الداخلي بالمنطقة 4 في الشكل (8-8).

يمكننا تجميع الفراغات الخارجية مع بعضها بالضغط compaction، لتكون فراغ خارجي واحد كبير يمكن الإستفادة منه.

ملحوظة:

- الفراغات الخارجية يمكن ضغطها لتكوين فراغ داخلي كبير .
- الفراغات الداخلية فلا يمكن ضغطتها وتجميعها كفراغ واحد.

- الفراغات الداخلية تتكون في التجزئة الثابتة.
- الفراغات الخارجية تتنتج في التجزئة الديناميكية.
- الضغط يمكن تطبيقيه في التجزئية الديناميكية.

مثال (4-8)

ذاكرة ديناميكية بمساحة حجمها 896 كيلوبايت. بعد تحميل ثلاث عمليات فيها بالاحجام 320، 224، و 288 سيكون الجزء الحر بها هو 64 كيلوبايت، كما في الشكل رقم (8-10) (د)، حيث سنحسبه كالآتي:

المساحة الحرة – مجموع حجم العمليات الثلاث المحملة =

896 – (320+224+288) = 64 كيلوبايت.

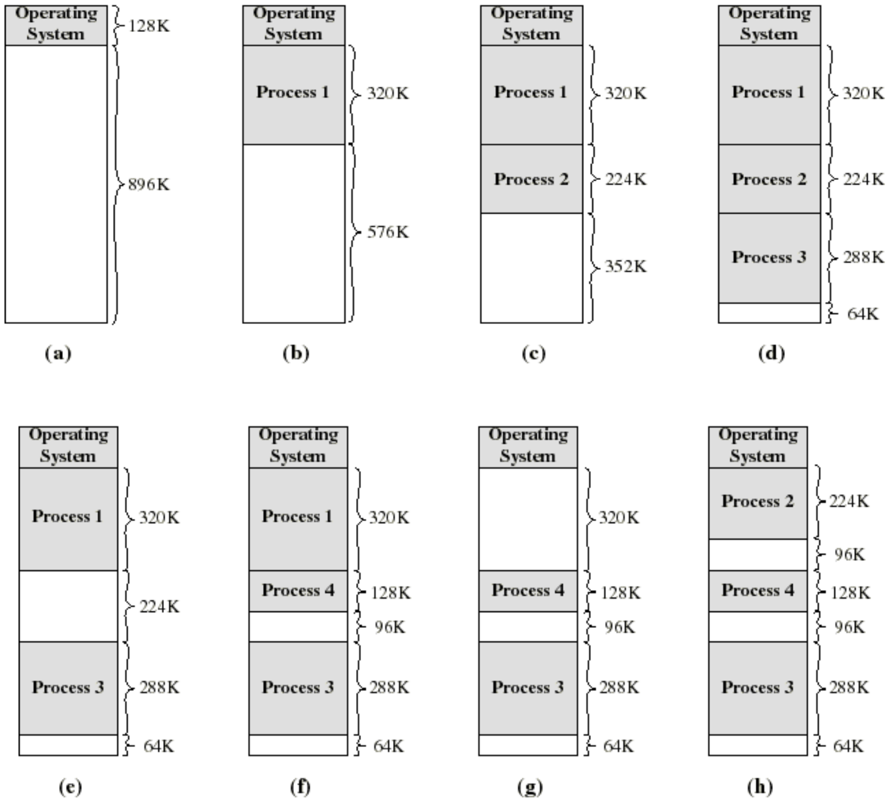
الآن إذا أردنا تحميل عملية رابعة بحجم 128 كيلوبايت، سيجيب نظام التشغيل بأنه لاتوجد مساحة كافية بالذاكرة، ذلك لأن 128 > 64. لذلك علي العملية الرابعة الانتظار حتى تتوفر مساحة كافية لها.

إذا أخرج نظام التشغيل العملية الثانية، يمكن للعملية الرابعة أن تحمل. لكن ستظهر فراغات (fragmentation) كما في الشكل (8-10) (و). إذا اخرج نظام التشغيل العملية الأولى، ودخلت مكانها العملية الثانية مرة أخرى سينشأ فراغ آخر كما في الشكل (8-10) (ح).

سؤال (8-1)

هل يمكن تحميل عملية حجمها 250k في الشكل (8-10) h؟

نعم إذا قمنا بضغط الفراغات الخارجية، حيث سيكون ذلك فراغ بحجم : 96+96+64=256K وهو كافي لتحميل العملية



شكل رقم (8-10): الفراغات.

سؤال (8-2)

هل يمكن تحميل عملية حجمها 250k في الشكل (8-11)؟ لا، لأن الفراغات داخلية ولا يمكن ضغطها.



شكل ر قم (8-11): ذاكرة بالتجزئية الثابتة.

8.9.2. تجميع الفراغات (defrag) أو الضغط (compaction)

إذا لم تتوفر منطقة لتحميل برنامج معين (لأنه أكبر من أي منطقة فارغة)، قد تستخدم بعض إدارات الذاكرة عملية الضغط أو تجميع الفراغات لتوفير مساحة أكبر وذلك بحساب مجموع الفراغات المتوفرة، فإذا كان حجمها أكبر أو يساوي حجم البرنامج تجمع هذه الفراغات لتكون فراغ واحد كبير يستطيع تحميل البرنامج. مثلا في الشكل (8-10) (h)، إذا استخدمنا الضغط سنجد أن الفراغ الذي سيجمع هو 96+96+64=256K.

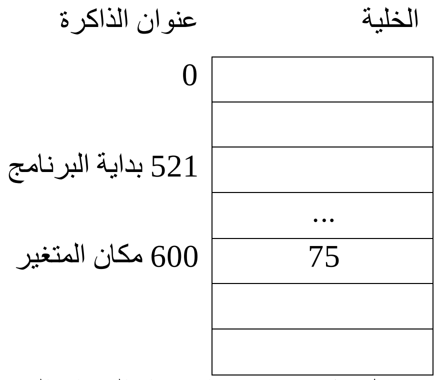
8.10. مشاكل تعدد المهام

الذاكرة متعددة المهام سواء كانت بالتجزئية الثابتة أو التجزئية الديناميكية أو غيرها، بها مشاكل ناتجة عن تعدد البرامج (أي وجود أكثر من برنامج بالذاكرة)، هذه المشاكل سنوضحها بالتفصيل أدناه ونبين كيف تتم معالجتها.

8.10.1. مشكلة تغيير الموقع Relocation

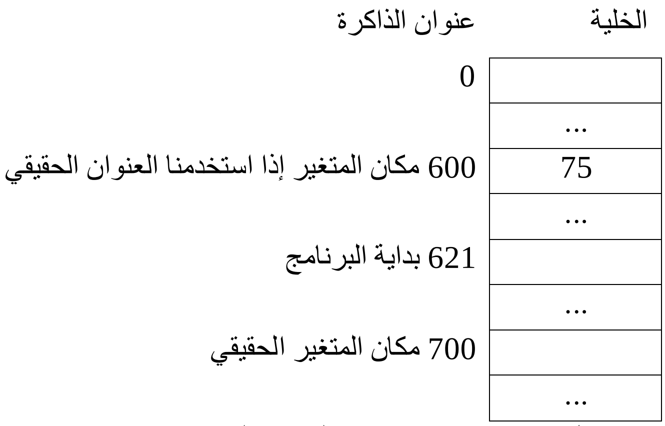
عند تحميل برنامج ما للتنفيذ، لا نعرف في أي منطقة بالذاكرة سيتحمل، أو إذا حدث تبديل (swap) أي تم إخراج البرنامج من الذاكرة لسبب ما، ثم تم إرجاعه مرة ثانية إلى الذاكرة، قد يرجع البرنامج في منطقة أخرى بالذاكرة غير التي كان فيها. عدم

معرفة مكان تحميل البرنامج بالذاكرة يجعل التعامل مع المتغيرات والدوال المرتبطة بعناوين الذاكرة غير ممكن، لأننا لا نعرف أين ستكون متغيراتنا ودوالنا بالذاكرة. مثلا إذا تحمل برنامج بموقع يبدأ بالعنوان 521، وكان هنالك متغير داخل البرنامج بالموقع 600، وأراد البرنامج تخزين قيمة ما داخل هذا المتغير فإنه سيستخدم العنوان 600 للتعامل مع المتغير، شكل رقم (8-12).



شكل رقم (8-12): استخدام العنوان الفيزيائي.

الآن إذا تم تحميل البرنامج في موقع آخر يبدأ بالعنوان 621 مثلا، الشكل (8-13). فإن البرنامج سيستخدم عنوان الذاكرة 600 للوصول للمتغير، ولكن العنوان 600 الآن أصبح خارج منطقة البرنامج.



شكل رقم (8-13): استخدام العنوان الحقيقي

أحد الحلول لهذه المشكلة هو تعديل أوامر البرنامج قبل تحميله بالذاكرة، حيث نضيف رقم بداية عنوان المنطقة التي سيحمل فيها البرنامج إلى كل أوامر البرنامج.

مثلا إذا تحمل البرنامج بمنطقة تبدأ بالعنوان 700، فكل أمر سيضاف إليه الرقم 700، فإذا كان البرنامج يتعامل مع متغير بالعنوان 100 فسيتغير العنوان إلى 700+100. بهذه الطريقة تحل مشكلة تغيير الموقع.

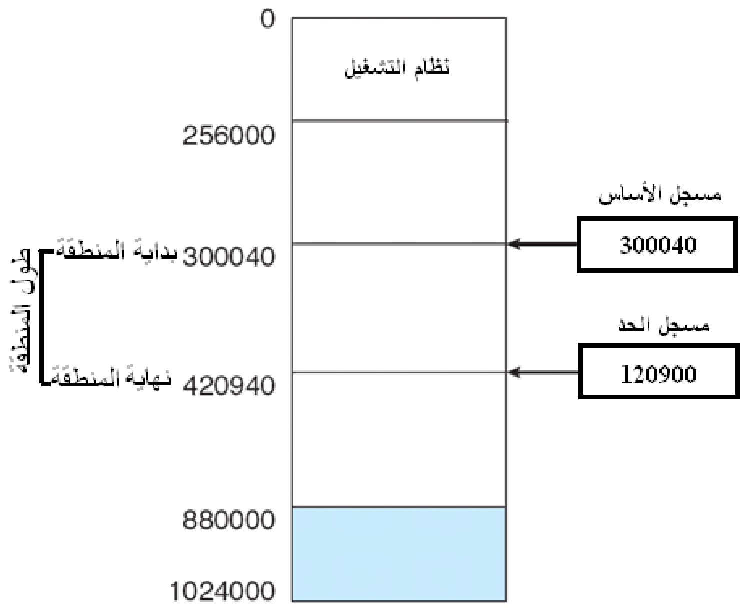
8.10.2. مشكلة الحماية Protection

وصول برنامج إلى منطقة برنامج آخر يسبب مشكلة تداخل قد تؤدي إلى تنفيذ خاطئ لهذه البرامج. فمثلا قد تستخدم برامج تخريبية (malicious program) هذه المشكلة كثغرة للوصول للبرامج والبيانات الأخرى (وذلك بإنشاء أوامر تمكنها من القفز (jump) إلى أي مكان بالذاكرة). وبما أننا نستخدم عناوين حقيقية، فليس هنالك طريقة لتوقيف مثل هذه البرامج من الوصول إلى أي خلية والكتابة أو القراءة منها.

8.10.3. حل مشكلتي تغيير المواقع والحماية

هنالك حل يمكن استخدامه لمعالجة مشكلتي إعادة تغيير المواقع والحماية في آن واحد، ألا وهو استخدام مسجلي الأساس (base) والطول (limit).

عند تحميل أي برنامج سيحتوي مسجل الأساس على عنوان بداية المنطقة التي سيتحمل بها، ومسجل الطول سيخزن به طول هذا المنطقة، الشكل (8-14).



شكل رقم (8-14): استخدام مسجلي الأساس والحد.

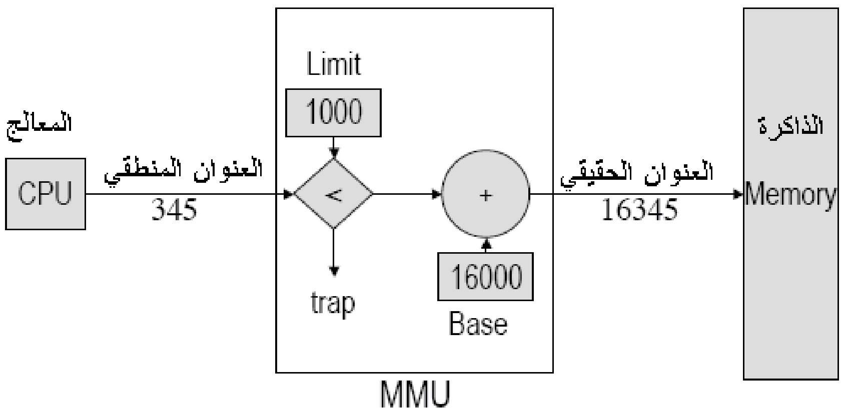
يستخدم المعالج عناوين منطقية (ترقيم تسلسلي من 0 إلى طول الحد) للتعامل مع البرنامج، وسيتم إضافة محتوى مسجل الأساس (عنوان بداية المنطقة) إلى العنوان المنطقي قبل إرساله للذاكرة. مثلا إذا أردنا تنفيذ أي أمر يتعامل مع الذاكرة، مثل الأمر (Load 345) ، فإنه لابد من إجراء الخطوات التالية قبل تنفيذ:

- هل الرقم 345 أكبر من طول المنطقة (limit)؟ (للحماية)
- إذا كانت الإجابة نعم سيرسل إشعار بأن العنوان خطأ (trap: address error).
- إذا كانت الإجابة لا سنحول العنوان المنطقي إلى عنوان حقيقي وذلك بإضافة محتوى مسجل الأساس للعنوان الذي يستخدمه الأمر:

Load (123+16000)

ثم يرسل إلى الذاكرة، الشكل (8-15).

من عيوب هذه الطريقة أنه لابد من إجراء الخطوات أعلاه على كل أمر يتعامل مع الذاكرة مما يتسبب في أبطاء التنفيذ.



شكل رقم (8-15): تحويل العنوان المنطقي إلى حقيقي.

8.11. العناوين المنطقية (***Logical addresses***)

تتكون الذاكرة من خلايا مصطفة وراء بعضها البعض وتأخذ كل خلية عنوان غير متكرر يسمى العنوان الحقيقي (تسمى أحيانا عناوين فيزيائية (physical addresses))، هذه العناوين هي التي تستخدمها الذاكرة. أما المعالج والبرامج فيستخدم عناوين منطقية (تسمى أحيانا عناوين ظاهرية (virtual addresses)). هذه العناوين تبدأ من الصفر وتزيد تسلسليا إلى نهاية البرنامج أو نهاية المنطقة (179 مثلا)، الشكل (8-16).

عنوان الذاكرة	الخلية	العنوان
000		
	...	
600	75	
	...	
621 بداية المنطقة		000
622		001
623		002
624		003
625		004
...	...	...
797		176
798		177
799		178
800 نهاية المنطقة		179

شكل رقم (8-16): العناوين المنطقية و العناوين الحقيقية.

السؤال هنا من يقوم بعملية تحويل العنوان ؟ الإجابة: وحدة إدارة الذاكرة (Memory Management Unit (MMU)).

8.11.1. ما هي وحدة إدارة الذاكرة (MMU) ؟

هي عبارة عن جهاز صغير (عتاد) موجود بالمعالج يقوم بتحويل العناوين المنطقية التي يستخدمها المعالج إلى عناوين حقيقية، بحيث لا يهتم ولا يعرف المعالج كيف يتم ذلك، فالمعالج يقوم بإرسال عنوانه المنطقي إلى MMU (مثلا العنوان رقم 346)، فيقوم MMU بتحويل العنوان المنطقي إلى حقيقي ثم يرسله للذاكرة (العنوان 14346)، فتصل المعلومة للمعالج وهو لا يدري موقعها الأصلي بالذاكرة، الشكل (8-15).

8.11.2. مسجل الأساس

عند تحميل البرنامج نضع عنوان بداية البرنامج في مسجل يسمى مسجل الأساس (base register)، هذا المسجل يستخدمه MMU مع العنوان المنطقي (تسمى أحيانا الإزاحة) لتوليد العنوان الحقيقي.

مثلا في الشكل (8-16)، سيكون محتوى مسجل الأساس هو 621، ويستخدم المعالج العناوين المنطقية من صفر إلى 179. أما MMU فيمكنه توليد أي عنوان فيزيائي بمجمع العنوان المنطقي إلى محتوى مسجل الأساس.

تنبيه:

- العنوان المنطقي يبدأ من الصفر إلى مسجل الحد-1
- العنوان الحقيقي: يبدأ من مسجل الأساس إلى مسجل الأساس+مسجل الحد-1
- هل العنوان المنطقي الذي تساوي قيمته قيمة مسجل الطول يعتبر عنوان صحيح؟ لا، لأن أكبر عنوان منطقي يجب أن يكون أقل من مسجل الحد بواحد.

قارن العناوين المنطقية مع العناوين الفيزيائية بالشكل (8-16) وذلك بجمع 621 للعنوان المنطقي وستجده يساوي العنوان الحقيقي لنفس الموقع.

الأمثلة التالية توضح كيف يقوم MMU بعملية التحويل.

مثال(8-5)

بالرجوع للشكل (8-15)، وضح كيف نحول العناوين المنطقية التالية إلى حقيقية:

- 4
- 177
- 200

الحل

من الشكل (8-15) نجد أن محتوى مسجل الأساس (621) وسيخزن في مسجل الطول طول المنطقة (179). للتحويل سيقوم MMU بمقارنة العناوين المنطقية مع محتوى مسجل الحد، فإذا كانت أصغر سيقوم بإضافة محتوى مسجل الأساس إلى العنوان المنطقي فينتج العنوان الحقيقي وعليه ستكون النتيجة كالتالي:

- 625 = 621 + 4
- 798 = 621 + 177
- خطأ، العنوان المنطقي أكبر من محتوى مسجل الحد.

مثال (8-6)

حول العناوين الحقيقية التالية إلى منطقية:

- 625
- 798
- 1000

سنقوم بطرح مسجل الأساس من العنوان الحقيقي فينتج عنه عنوان منطقي، وذلك كما يلي:

- 4 = 621 – 625
- 177 = 621 – 798

- 1000 – 621 = 279 (خطأ لأن العنوان الحقيقي خارج حد العملية).

8.12. الذاكرة بالصفحات (***Paging***)

نلاحظ أنه إذا قسمنا البرنامج لجزأين كبيرين فإن:

- يتطلب كل جزء فراغ كبير لتحميله به.
- المبادلة ستكون بطيئة جدا وذلك لكبر حجم الأجزاء المتبادلة.
- إذا احتجنا إلى جزئية صغيرة من النصف الذي بالقرص فسنضطر إلى تبديله (كله) مع الجزء الذي بالذاكرة، بعد تنفيذ الجزئية الصغيرة المطلوبة من هذا الجزء وأردنا الرجوع لإكمال تنفيذ الجزء الأول الذي تم إخراجهن فسنضطر لتبديل الجزأين مرة أخرى، مما يتسبب في أبطاء العمل بصورة واضحة، ذلك لأن التعامل مع القرص الصلب يؤثر تأثيرا كبيرا في الأداء.

هدفنا هنا إذن هو تحاشي التبديل بقدر المستطاع (تقليل التعامل مع القرص ما أمكن ذلك)، والإستفادة من أي مساحة فارغة بالذاكرة (أي ليس من الضروي أن تكون المساحة المتاحة في الذاكرة كافية لتحميل كل البرنامج أو جزء كبير منه حتى ننفذه) والحل استخدام الصفحات.

8.12.1. الصفحات

تقسيم البرنامج إلى أجزاء صغيرة متساوية في الحجم تسمى صفحات (pages)، وتقسم الذاكرة إلى مناطق صغيرة متساوية في الحجم (ومساوية لحجم الصفحة) تسمى إطارات (frames). بحيث يكون كل إطار قادرا على تخزين صفحة، شكل (9-2).

لتنفيذ برنامج بعدد ن صفحة، فسنحتاج إلى ن إطار فارغ بالذاكرة، وإلا سنحتاج إلى ذاكرة ظاهرية.

قد تنشأ فراغات داخلية في Internal fragmentation في ذاكرة الصفحات. مثلا إذا كان حجم الصفحة 4kb وكان لدينا برنامج حجمه 110kb ، فإننا سنقسم البرنامج كالتالي:

27 = 110/4 الباقي 2، إذن سأحتاج إلى 27 صفحة بالإضافة إلى الباقي من البرنامج وهو 2kb فأضطر إلى وضعها في صفحة (وبما أن الصفحة حجمها أربعة فإن الباقي وهو 2kb سنضعه في صفحة ويكون لدينا فراغ داخلي حجمه 2kb)، لذلك سنحتاج إلى 28 صفحة للبرنامج مع فراغ داخلي حجمه 2kb في الصفحة الأخيرة.

8.12.2. التعامل مع العناوين في الصفحات

يتكون العنوان المنطقي من شقين:

- عنوان الصفحة (رقم الصفحة).
- عنوان الخانة داخل الصفحة (الإزاحة offset).

رقم الصفحة	الإزاحة
p	*d*

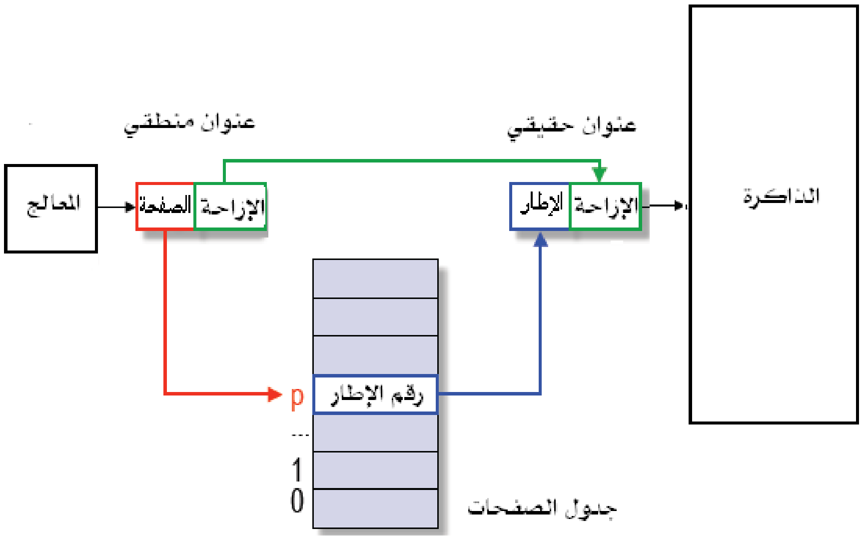
شكل رقم (8-17): العنوان المنطقي.

فإذا كان p=7، و d=20، فهذا يعني أننا نريد الخانة رقم 21 (لأن أول خانة في الصفحة رقمها هو صفر) بالصفحة السابعة.

ويتكون العنوان الحقيقي من شقين هما:

- عنوان الإطار الذي تخزن به الصفحة في الذاكرة.
- عنوان الخانة داخل الإطار (الإزاحة).

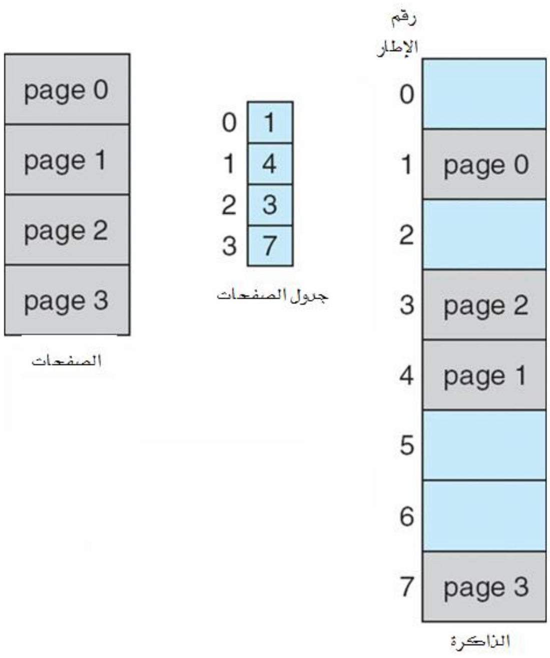
عملية تحويل العنوان المنطقي إلى حقيقي يتم بجهاز (hardware) يستقبل العناوين المنطقية من المعالج ويحولها إلى حقيقية قبل إرسالها للذاكرة. تعتمد عملية التحويل هذه على جداول الصفحات.



شكل رقم (8-18): جهاز تحويل العنوان منطقي إلى حقيقي.

في الشكل (8-18)، يستخدم المعالج عنوان الصفحة ومكان الخانة داخل الصفحة (الإزاحة)، فيقوم الجهاز بالبحث عن رقم الإطار الذي وضعت به الصفحة بالذاكرة من جدول الصفحات، حيث يمثل عنوان الصفحة فهرس نصل به إلى رقم الإطار الذي تخزن به الصفحة في الذاكرة. ندمج رقم الإطار مع الإزاحة فينتج العنوان الحقيقي

يتم إنشاء جدول صفحات لكل عملية بحيث يحتوي هذا الجدول على رقم الصفحات التي تم تحميلها بالذاكرة وأرقام الإطارات التي وضعت فيها هذه الصفحات، كما في الشكل (8-19).



شكل رقم (8-19): جدول الصفحات يوضح مكان الصفحات بالذاكرة.

مثال (8-7)

حول العناوين المنطقية التالية إلى عناوين حقيقية (بالرجوع للشكل (9-5)):

2	10	1	0	3	3	0	7

الحل

سنبحث عن الإطار الذي توجد به الصفحة من جدول الصفحات ونستبدله برقم الصفحة (ونضع الغزاحة كما هي) فتكون العناوين الحقيقية كما يلي:

3	10	4	0	7	3	1	7

مثال (8-8)

الشكل التالي يوضح البيانات التي توجد داخل الصفحات، ومكان تواجد هذه الصفحات بالذاكرة. المطلوب هنا معرفة مكان معلومة معينة داخل الصفحة مثلا الحرف a أين يوجد بالذاكرة ؟

الحل

الحرف a يوجد الصغحة الاولى (رقم صفر)، ويوجد في أول خانة داخل الصفحة (الإزاحة=صفر) ، سنقوم بتحويل رقم الصفحة إلى رقم الإطار من جدول الصفحات حيث سنجد أن الصفحة رقم صفر توجد في الإطار رقم خمسة، لنصل للمعلومة مباشرة نضرب رقم الإطار في طول الصفحة (4) ونضيف إليه الإزاحة:

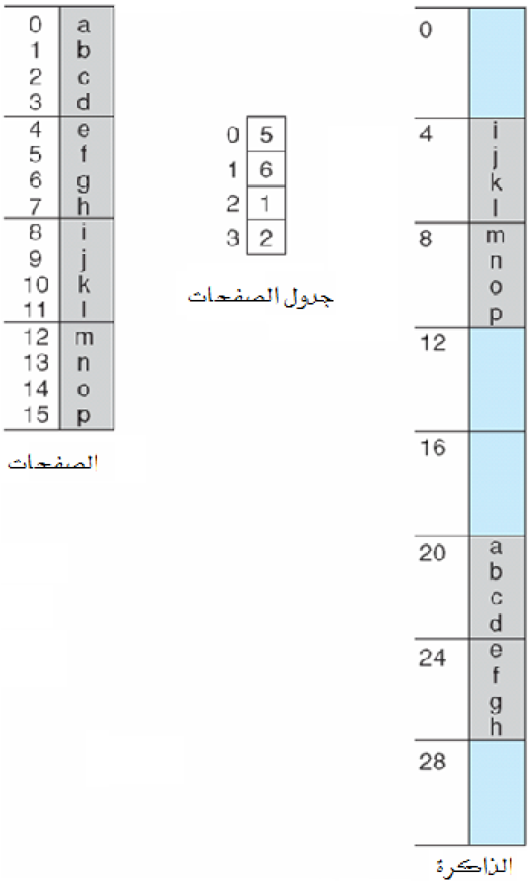
$$5 \times 4 + 0 = 20$$

فنجد أن العنوان رقم 20 بالذاكرة يحتوي فعلا على الحرف a.

أيضا الحرف p مثلا في الصفحة رقم 3، في الخانة رقم 3، الصفحة 3 توجد بالإطار 2 (من جدول الصفحات)، لمعرفة مكان الحرف p بالذاكرة سنجري العمليية التالية:

$$2 \times 4 + 3 = 11$$

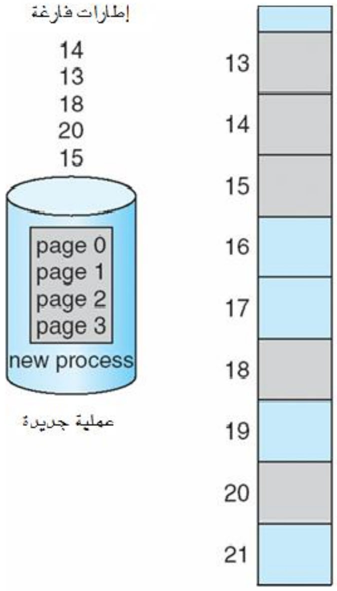
ونجد 11 هي مكان الحرف p بالذاكرة.



شكل رقم (8-20): مثال لصفحات بالذاكرة مع جدول الصفحات.

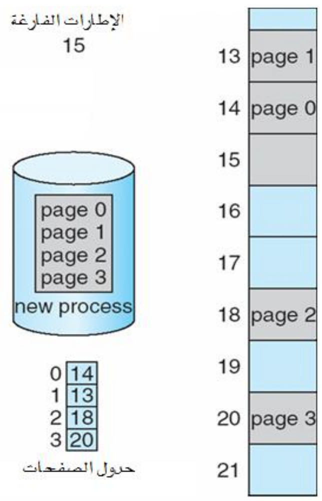
مثال (8-9)

إذا كان لدي عملية تتكون من 4 صفحات ولدي إطارات فراغة كما موضح بالشكل، إنشي جدول الصفحات بعد تحميل العملية بالذاكرة.



شكل رقم (8-21): عملية جديدة نريد تحميلها في الذاكرة.

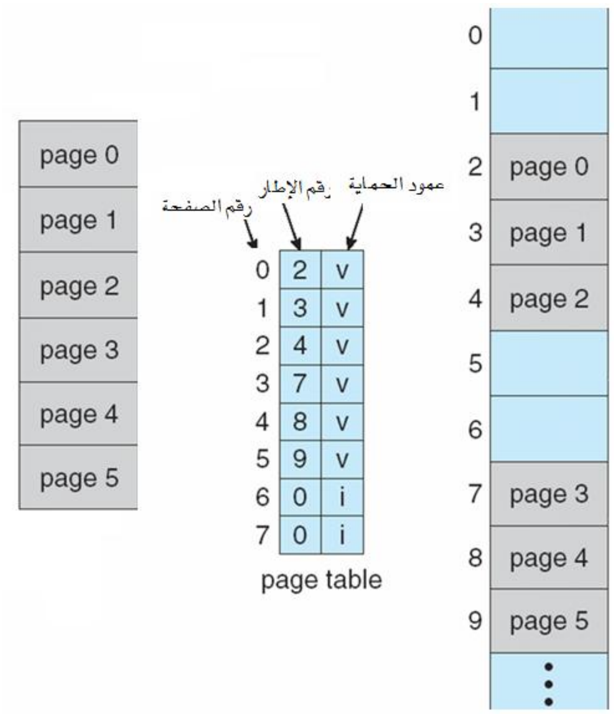
الحل



شكل رقم (8-22): تحميل عملية بالذاكرة.

8.12.3. حماية الذاكرة

كيف أعرف أن صفحة معينة تنتمي إلى عملية معينة ؟ يمكن تخصيص خانة (bit) في كل إطار لهذا الغرض، بحيث تحدد قيمة هذه الخانة هل تنتمي الصفحة للعملية أم لا. فإذا كانت قيمة البت هي v وتعني (valid) فهذا يعني أن الصفحة تنتمي للعملية، اما إذا كانت قيمة البت i وتعني (in-valid)، فهذا يخبرنا بأن الصفحة لا تنتمي للعملية. يكون هنالك عمود حماية لجدول الصفحات كما في الشكل (8-23).



شكل رقم (8-23): عمود الحماية يضاف إلى جدول الصفحات.

من الشكل (8-23) نجد أن الصفحات الصحيحة هي 0,1,2,3,4,5 والتي تظهر في اليسار، لذلك نجد أمامها v في جدول الصفحات، أما الصفحات 5,7 فهي غير صحيحة لذلك نجد أماها i وفي رقم الإطار صفر.

8.12.4. الصفحات المشتركة Shared Pages

يمكن جعل الصفحات المتشابهة بين عدة عمليات مشتركة في الذاكرة لتوفير مساحة بالذاكرة وإزالة التكرار، فبدلا من وضع نسخة من هذه الصفحات لكل عملية

نضع نسخة واحدة ونجعلها مشتركة بين العمليات. أما الصفحات التي تخص كل عملية فتكون غير مشتركة.

8.12.4.1. الشفرات المشتركة Shared code

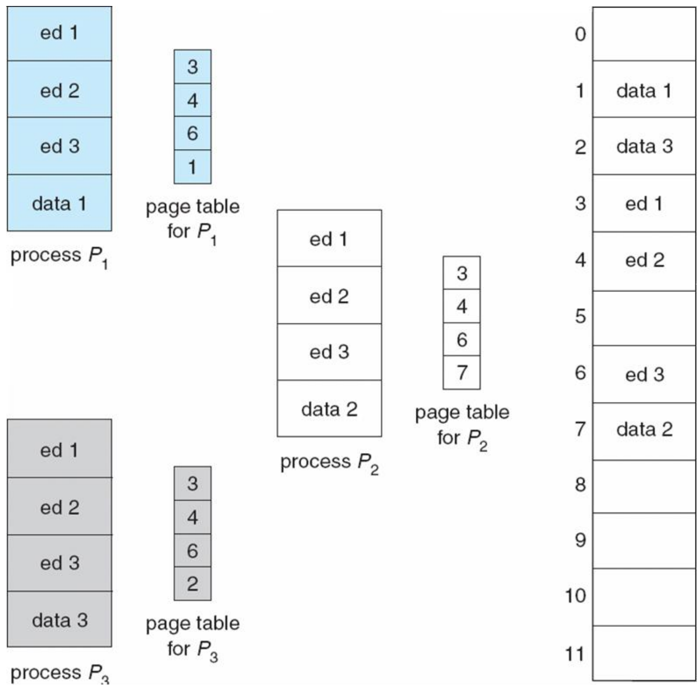
نسخة واحدة للقراءة فقط لكل العمليات مثل محرر النصوص، المترجمات.

الشفرة المشتركة يجب أن تكون في نفس المكان في مجال العنوان المنطقي logical address space لكل العمليات، مثلا الشكل (8-24) نجد عنوان ed متشابه في كل العمليات (3،4،6).

8.14.4.2. الشفرات والبيانات الخاصة Private code and data

كل عملية يكون لديها نسخة منفصلة لشفراتها وبياناتها.

الصفحات التي تحتوي شفرات وبيانات خاصة يمكن أن تكون في أي مكان في مجال العنوان المنطقيlogical address space، مثلا في الشكل (8-24) نجد أن لكل عملية عنوان مختلف لبياناتها الخاصة (p1=1, p3=2, p2=7).



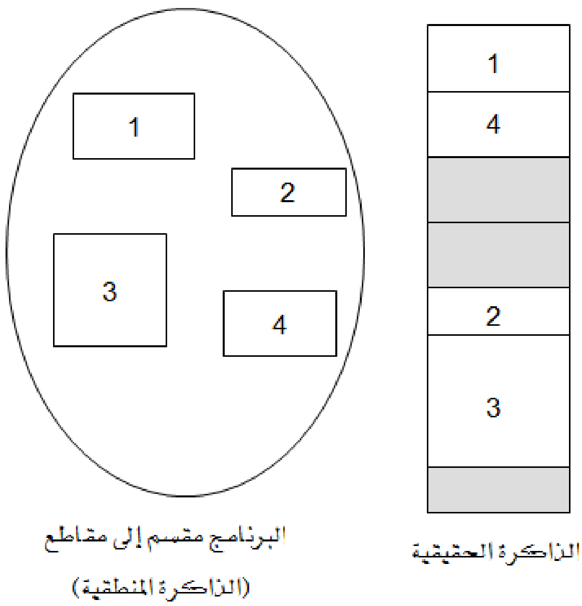
شكل رقم (8-24): صفحات مشتركة بين عدة عمليات.

8.13. التقطيع (*segementation*)

هو طريقة لإدارة الذاكرة تدعم نظرة المستخدم للذاكرة، حيث نقوم بتقسيم البرنامج إلى أجزاء منطقية، فالبرنامج الرئيسي يكون في مقطع، الدوال في مقطع والبيانات في مقطع آخر، وهكذا.

يعتبر المقطع وحدة منطقية مثل البرنامج الرئيسي، دالة معينة، كائن، المتغيرات العامة والخاصة، المكدس (stack)، المصفوفات.

بعد تقسيم البرنامج إلى مقاطع نضع كل مقطع في خانة بالذاكرة، الشكل (8-25).



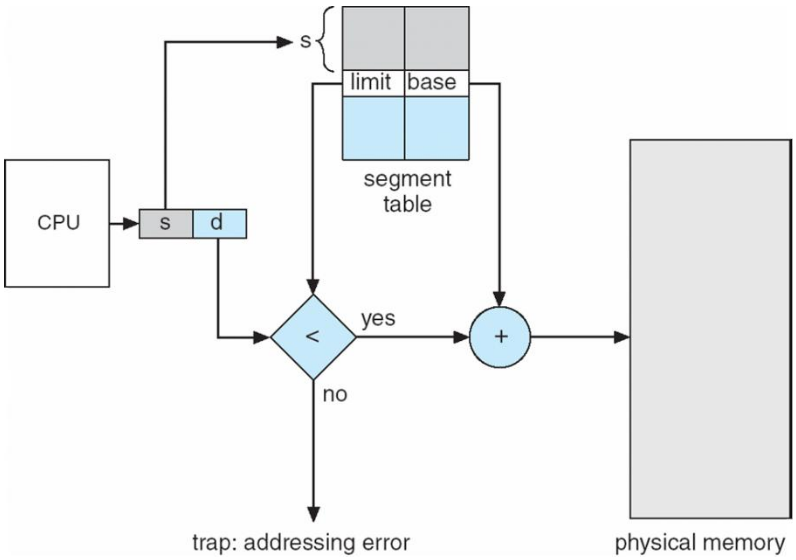
شكل رقم (8-25): تحميل مقاطع البرنامج في الذاكرة.

8.13.1. العناوين في التقطيع

يتكون العنوان المنطقي من : رقم المقطع، ورقم الإزاحة (segment-number،offset).

تتم عملية التحويل بين العنوان المنطقي والعنوان الحقيقي استخدام مسجلي أساس وحد، حيث يحتوي مسجل الأساس عنوان بداية المقطع في الذاكرة ومسجل الطول يحتوي على طول المقطع (limit). توضع مسجلات الأساس والطول لكل المقاطع في جدول واحد يسمى جدول المقاطع (segemet table)، حيث هنالك جدول مقاطع لكل عملية.

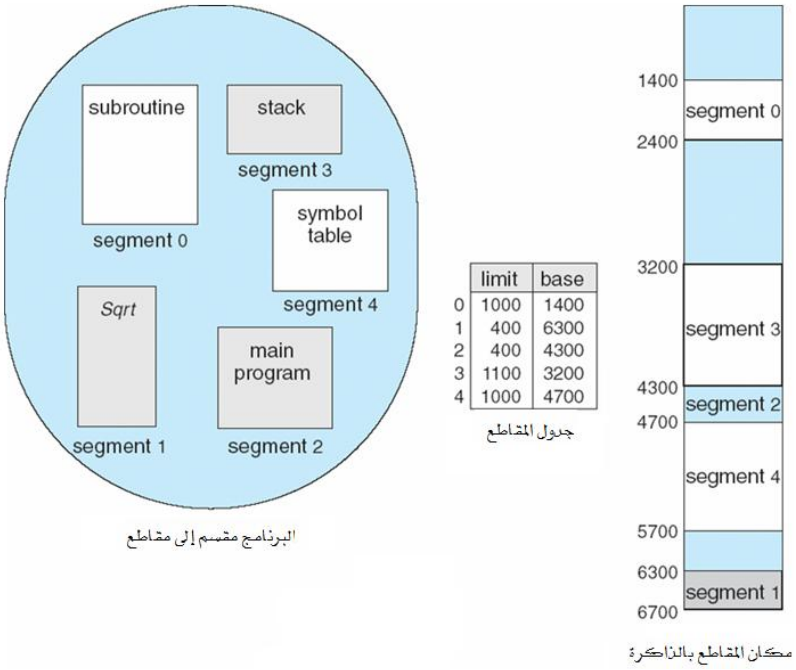
هنالك جهاز يقوم بعملية تحويل العناوين من منطقية إلى حقيقية يعمل كما في الشكل (8-26).



شكل رقم (8-26): تحويل العنوان المنطقي إلى حقيقي.

مثال (8-9)

الشكل (8-27) يوضح برنامج مكون من 5 مقاطع، ويبين جدول المقاطع مكان كل مقطع بالذاكرة الرئيسية.



شكل رقم (8-27): مثال لمقاطع برنامج في الذاكرة مع جدول المقاطع.

لتوضيح كيف يتم التحويل من العنوان المنطقي إلى الحقيقي، لنحول العنوان المنطقي التالي إلى حقيقي:

0	**20**

هذا العنوان يعني أنني أريد معلومة من المقطع رقم 0 الإزاحة رقم 20.

الحل

- أبحث في جدول المقاطع عن قيمة مسجل الأساس للمقطع 20 وهي 1400.
- قارن قيمة الإزاحة مع مسجل الطول (أي هل 20 أصغر من 1000) ؟ إذا كانت الإجابة نعم فسأقوم بالتحول وإلا سيكون هنالك خطأ في الإزاحة.

- إذن العنوان الحقيقي سيكون محتوى مسجل الأساس + الإزاحة أي 1420 =20+1400

مثال (8-10)

حول العنوان المنطقي التالي إلى حقيقي :

0	1001

الحل

- مسجل الأساس للمقطع هو 1400
- مسجل الطول للمقطع هو 1000
- عند مقارنة مسجل الطول مع الإزاحة نجد ان الإزاحة أكبر من مسجل الطول وهذا يعني أننا سنكون خارج المقطع (وصول خاطئ).

هذا يعني أننا لا نستطيع تحويل هذا العنوان لأنه عنوان خطأ.

8.14. خلاصة

تحدثنا في هذا الباب أهداف إدارة الذاكرة، وعن أنواع إدارات الذاكرة وهي الأحادية،والتجزئية الثابتة والتجزئية الديناميكية والصفحات والتقطيع. حيث تعتبر الذاكرة الأحادية قديمة وغير مستخدمة حاليا، والتجزئية الثابتة تولد فراغات داخلية لا يمكن الإستفادة منهان بينما تولد التجزئية الديناميكية فراغات خارجية يمكن ضغطتها للإستفادة منها. الذاكرة بالتصفح تساعد في الاستفادة من فراغات الذاكرة الصغيرة حيث يمكن تحميل صفحات في فراغات صغيرة مبعثرة. التقطيع هو إدارة لتقسيم العملية إلى مقاطع ونضع كل مقطع في جزء من الاذاكرة. أيضا وضحنا أن العنوان المنطقي هو العنوان الذي يستخدمه المعالج والبرامج بينما تستخدم الذاكرة عناوين حقيقية. وسنتحدث عن إمكانية تنفيذ برامج أكبر من حجم الذاكرة الرئيسية فيما يسمى الذاكرة الظاهرية.

8.15. تمارين محلولة

1. إذا كان لدينا جدول المقاطع (segment table) التالي:

رقم المقطع	الأساس (base)	الحد/الطول (limit)
0	300	600
1	7000	20
2	0	300
3	1000	6000
4	900	100

حول العناوين المنطقية التالية إلى عناوين حقيقية :

- 0, 500
- 1, 10
- 2, 500
- 3, 2400
- 4, 99

أرسم شكل الذاكرة مع توضيح هل توجد فراغات خارجية أم لا ؟

الحل:

0,500 → 800

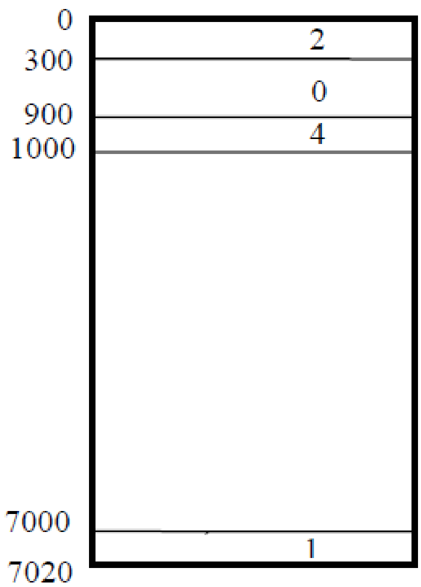
1,10 → 7010

2,500 → -

3,2400 → 3400

4,99 → 999

العنوان المنطقي 2,500 لا يمكن تحويله لأن 500 أكبر من طول المقطع (الحد هو 300).



من الشكل أعلاه نجد أنه لا يوجد فراغ خارجي.

8.16. تمارين غير محلولة

1. أذكر خمسة من أهداف إدارة الذاكرة ؟

2. ما الفرق بين العنوان المنطقي والعنوان الحقيقي ؟

3. أذكر مشكلتي تعدد المهام ، مع توضيح الحل لها ؟

4. ما الفرق بين الذاكرة بالتجزئية الثابتة والذاكرة الديناميكية ؟ و ماهي عيوب كل منهما ؟

5. ماهي الفراغات وما علاجها ؟

6. هنالك نوعين من الفراغات إذكرهما مع تبيين في أي نوع من إدارات الذاكرة تنشأ كل واحدة منها؟ وما الفرق الرئيسي بينهما ؟

7. ضع دائرة حول الإجابة/الإجابات الصحيحة (قد يكون هنالك أكثر من إجابة صحيحة)

7.1. ما الذي لا نرغب به في الذاكرة:

- سريعة
- كبيرة
- متطايرة
- غير متطايرة

7.2. إذا كان لدينا ذاكرة مقسمة إلى الأجزاء التالية:

100، 500، 200، 300، 600

وأردنا تحميل العمليات التالية فيها:

P1=5, p2=100, p3=300, p4=500, p5=200, P6=600

فإن :

- P6 ستنتظر في طريقة الأول ff

- P6 ستنتظر في طريقة الأنسب bf،
- P6 ستنتظر في طريقة الأسوأ wf

7.3. الجهاز الذي يقوم بتحويل العنوان المنطقي إلى عنوان حقيقي يسمى:

- CPU
- MPU
- MMU
- MNU

7.4. إذا كان لدي 3 فراغات داخلية، الفراغ الأول بحجم 100K، والفراغ الثاني بحجم 90K، والفراغ الثالث حجمه 110K، فهل يمكنني تحميل عملية حجمها 300K ، علما بأن مجموع هذه الفراغات يساوي 300K.

- نعم، إذا استخدمت الضغط.
- لا يمكن لأني لا استطيع ضغط الفراغات.
- الإجابتان خطأ.
- الإجابتان خطأ.

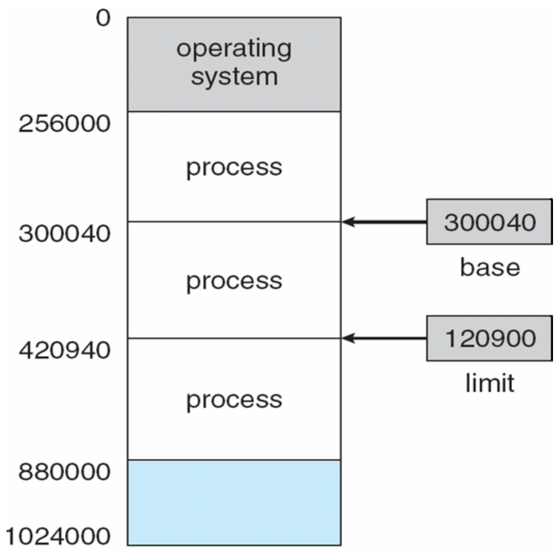
7.5. إذا كان لدينا برنامج حجمه 100 K وكان حجم الصفحة 3 K ، فإنه سيكون لدينا:

- 33 صفحة بدون فراغ داخلي في الصفحة الأخيرة
- 34 صفحة بفراغ داخلي في الصفحة الأخيرة
- 33 صفحة بفراغ داخلي في الصفحة الأخيرة
- 34 صفحة بدون فراغ داخلي في الصفحة الأخيرة

8. أجب بنعم أو لا (مع تصحيح الإجابة الخاطئة)

- يتعامل برنامج المستخدم مع العناوين الحقيقية ولا يرى أبدا العناوين المنطقية.
- الفراغات الداخلية لا يمكن ضغطتها.
- التجزئة الديناميكية تولد فراغات داخلية بينما التجزئة الثابتة تولد فراغات خارجية.

9. الرجوع للشكل (8-28)، أجب على الأسئلة التي تليه.



شكل رقم (8-28): الصفحات.

بإفتراض أن مسجل الأساس يحتوي على 300040، ومسجل الطول يحتوي على 120900، كما في الشكل أعلاه.

حول العناوين المنطقية التالية إلى عناوين حقيقية:

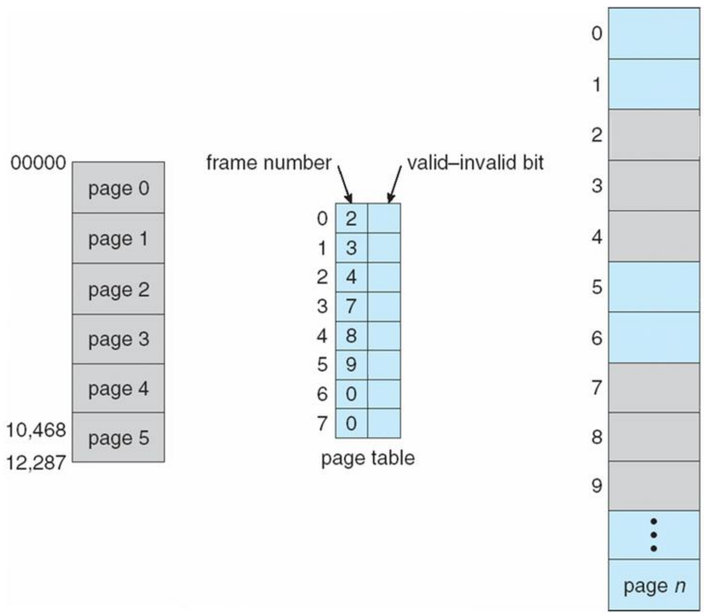
123456، 121، 1115

حول العناوين الحقيقية التالية إلى منطقية:

300040، 300039، 400040

10. ما هو مفهوم الصفحات ؟

11. أكمل الرسم أدناه بتوضيح أماكن تحميل الصفحات بالذاكرة وتوضيح الصفحات الصحيحة من الغير صحيحة في جدول الصفحات بوضح الحرف v أمام الصفحات الصحيحة والحرف i أمام الصفحات الغير صحيحة ؟



الباب التاسع:

الذاكرة الظاهرية

الباب التاسع

الذاكرة الظاهرية (Virtual Memory)

9.1. في الماضي

في الخمسينيات وقبل ظهور الذاكرة الظاهرية، كان كل برنامج نريد تنفيذه لابد من أن يكون في الذاكرة أولا، هذا يعني أن أي برنامج يجب أن يكون أقل أو يساوي حجم الذاكرة المتوفرة (الفارغة). تعتبر هذه مشكلة في البرامج الكبيرة، إذ لا يمكن تشغيل برامج حجمها اكبر من حجم الذاكرة. هذا يعني أننا لا نستطيع تشغيل برامج كثيرة بالذاكرة. البرامج كبيرة الحجم يجب تقسيمها إلى برامج صغيرة وتنفيذ كل برنامج على حدة، إذ لا يمكن تحميل أكثر من برنامج بالذاكرة. تشغيله برنامج كبير فلابد من شراء ذاكرة تتسع له، ولكن:

- ستكون مكلفة.
- مهما كانت الذاكرة كبيرة ستكون هنالك برامج أكبر.

هنالك أسماء متعددة تستخدم للذاكرة الظاهرية (virtual memory)، فقد يطلق عليها الذاكرة الافتراضية أو الذاكرة الخيالية أو الذاكرة الوهمية، كلمات كثيرة لمعنى واحد.

9.2. تعريف الذاكرة الظاهرية

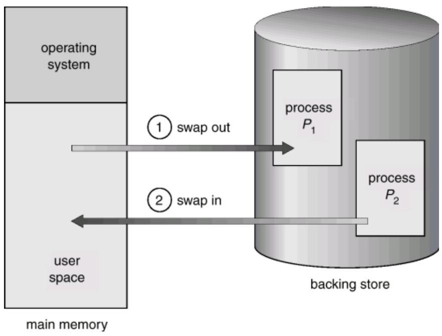
نظم التشغيل الحديثة تستخدم جزء من القرص الصلب كامتداد للذاكرة الرئيسية. حيث يحمل جزء من البرنامج في الذاكرة الرئيسية (إذا لم يكن هنالك متسع لتحميله كاملا في الذاكرة الرئيسية) وباقي أجزاء البرنامج تحمل في جزء من القرص الصلب. في هذه الحالة يعتبر هذا الجزء من القرص الصلب امتداداً للذاكرة الرئيسة ويسمى الذاكرة الظاهرية وبهذا تكون هذه النظم قد حلت مشكلة ارتباط حجم البرنامج بحجم المساحة المتوفرة من الذاكرة الرئيسية.

يبدأ المعالج في تنفيذ جزء البرنامج الموجود بالذاكرة، وإذا أحتاج إلى أوامر أو بيانات من الجزء الآخر (الموجود بالقرص) سيقوم مدير الذاكرة بتبديل الجزأين (swap).

9.3. المبادلة ***Swapping***

المبادلة هي تحويل البرنامج (أو جزء منه) من الذاكرة الرئيسية إلى القرص أو العكس. تتم المبادلة على مرحلتين، الشكل (9-1)، هما:

- تحويل البرنامج (أو جزء منه) من الذاكرة إلى القرص (swap out).
- تحميل البرنامج (أو جزء منه) من القرص إلى الذاكرة (swap in).



شكل رقم (9-1): المبادلة.

9.4. الذاكرة الظاهرية

تعتبر الذاكرة الظاهرية فصل بين ذاكرة المستخدم المنطقية والذاكرة الحقيقية وتختلف عن الذاكرة الحقيقية في الآتي:

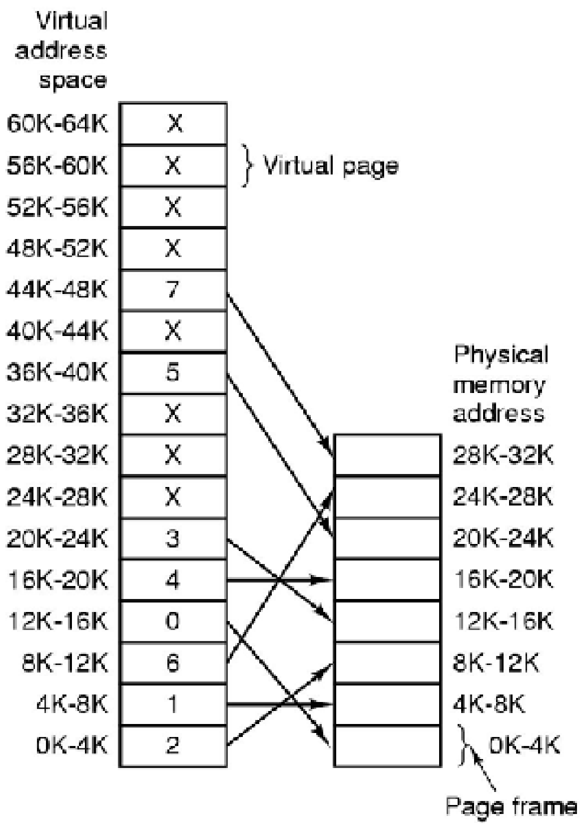
- فقط جزء من البرنامج يكون بالذاكرة.
- قد يكون مجال العنوان المنطقي أكبر من مجال العنوان الحقيقي.
- يمكن لعدة عمليات التشارك في مجال عناوين.

يمكن تطبيق الذاكرة الظاهرية في :

- إدارة الذاكرة بالصفحات.
- إدارة الذاكرة بالمقاطع.

9.5. الذاكرة الظاهرية في الصفحات

إذا كان صفحات البرنامج أقل أو تساوي الإطارات بالذاكرة فلن نحتاج إلى ذاكرة ظاهرية. ولكن نحتاج للذاكرة الظاهرية إذا كانت صفحات البرنامج أكبر من الإطارات المتوفرة بالذاكرة. مثلا في الشكل (9-2) نجد أن الذاكرة مقسمة إلى 8 إطارات، والبرنامج مقسم إلى 16 صفحة. سيكون هنالك 8 صفحات بالذاكرة من البرنامج (أمامها علامة X) والبقية ستكون في القرص الصلب (ذاكرة ظاهرية).



شكل رقم (9-2): الصفحات.

التعامل مع الذاكرة الظاهرية:

- إحضار الصفحة التي نحتاجها إلى الذاكرة.
- نبحث عن الصفحة المطلوبة المطلوبة في جدول الصفحات
  - إذا كانت الصفحة غير صحيحة ، نتوقف
  - إذا كانت ليست بالذاكرة، نحضرها إلى الذاكرة
- لا نحضر صفحة إلى الذاكرة ما لم نحتاجها

هنالك خوارزميات عديدة يستخدمها مدير الذاكرة ليحدد أي صفحة يخرج من الذاكرة عند ما نريد إطار فارغ لتخزين الصفحة المطلوبة به (فمن الصفحة الضحية)؟

قبل إخراج الصفحة (استخدام مكانها)، سيقوم مدير الذاكرة باختبار هل الصفحة المراد استخدام مكانها قد تم تعديلها أم لا ؟ إذا تم تعديلها يجب أن تحفظ بالقرص، أما إذا لم يتم تعديلها فنكتفي بنسختها القديمة الموجودة بالقرص.

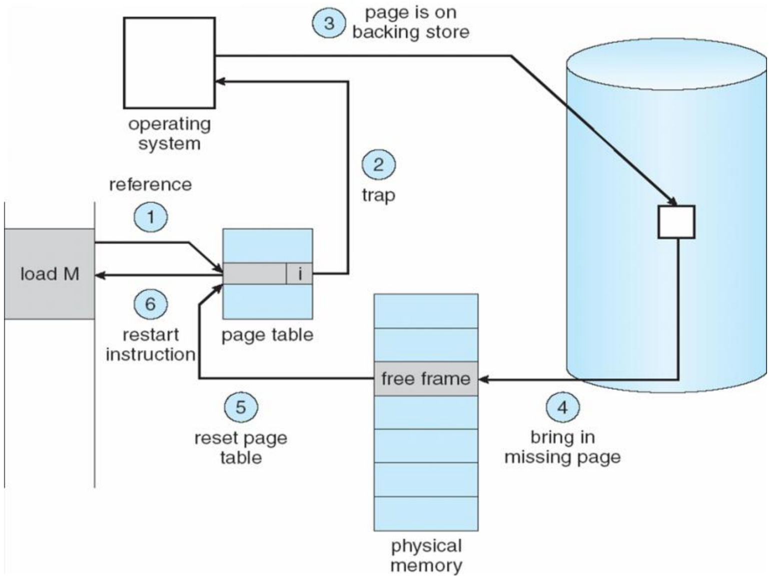
9.5.1. خطأ صفحة (page fault)

عندما يطلب برنامج صفحة غير موجودة بالذاكرة (إفتقاد صفحة)، سيسبب ذلك قفز إلى نظام التشغيل برسالة تقول أن هنالك خطأ صفحة page fault ، أي أن البرنامج يريد صفحة ولم يجدها بالذاكرة. على نظام التشغيل توفير الصفحة المفتقدة للبرنامج الذي إفتقدها حسب الخطوات التالية:

- إذا كانت الصفحة ليست صحيحة، يتوقف abort.
- إذا كانت الصفحة صحيحة لكنها ليست بالذاكرة:
  - الحصول على إطار فارغ.
  - وضع الصفحة المطلوبة في هذا الإطار الفارغ.
  - تحديث جداول الصفحات بوضع رقم الصفحة ورقم الإطار الذي وضع بها.

- o تعديل بت التصحيح إلى **v** ، وهذا يعني أن الصفحة صحيحة.

- إعادة تنفيذ الأمر الذي سبب خطأ صفحة page fault.



ماذا يحدث إذا لم نجد إطار فارغ ؟

إذا طلبنا صفحة غير موجودة بالذاكرة ولم يجد نظام التشغيل إطار فارغ لإحضار الصفحة المطلوبة فيه، ماذا يفعل نظام التشغيل ؟

سيقوم بعملية استبدال (swap)، أي إخراج صفحة من الذاكرة (نسميها الصفحة الضحية) وإدخال الصفحة المطلوبة مكانها.

هنا يظهر سؤال هام وهو أي صفحة سيخرج من الصفحات التي بالذاكرة (من ستكون الصفحة الضحية) ؟ هنالك عدة خوارزميات تحدد أي صفحة سنخرج.

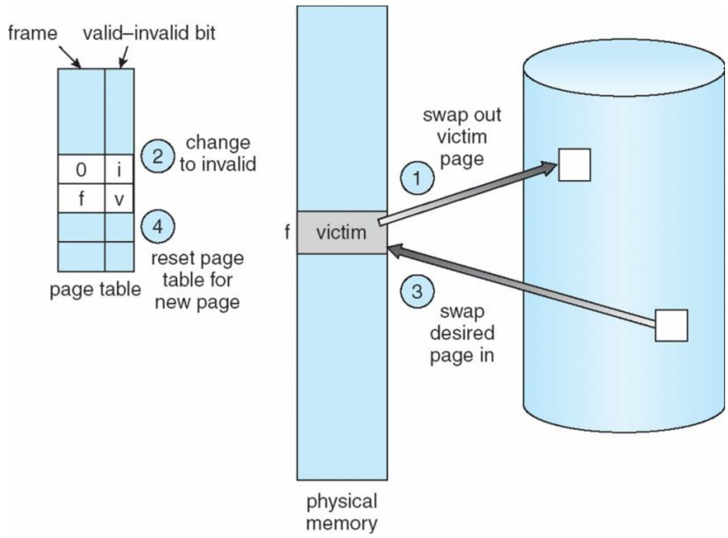
يستخدم نظام التشغيل أحد هذه الخوارزميات، مع وضع إعتبار للأداء في اختيار الخوارزمية (نريد خوارزمية تخرج صفحة لا نحتاجها في القريب العاجل وهذا يقلل عدد أخطاء الصفحات page faults).

9.5.2. استبدال الصفحات Page Replacement

نستفيد من استبدال الصفحات في استخدام إطارات قليلة لتنفيذ عملية بصفحات كثيرة، وهي الفكرة الرئيسية في الذاكرة الظاهرية.

خطوات استبدال الصفحة:

1. أبحث عن موقع الصفحة في القرص
2. أبحث عن إطار فارغ، إذا وجد إطار نستخدمه.
3. إذا لم يوجد إطار فارغ، نستخدم خوارزمية لإختيار الإطار الضحية **victim** frame.
4. إخراج الصفحة الضحية من الإطار الضحية.
5. إحضار الصفحة المطلوبة إلى الإطار الضحية.
6. تعديل جدول الصفحات.
7. إعادة تنفيذ الأمر الذي سبب خطأ الصفحة.



9.6. خوارزميات استبدال الصفحات

Page Replacement Algorithms

هنالك العديد من الخوارزميات التي تستخدم في استبدال الصفحات منها:

- إخراج الصفحة الأقدم (الأول أولا تخرج أولا)، first come first out (fifo)
- إخراج الصفحة التي لا نحتاج لها قريبا (الخوارزمية المثلى optimal)
- خوارزمية الصفحة الاقل استخدام LRU
- خوارزمية LRU التقريبية approxmination
- خوارزمية الساعة.
- خوارزمية الفرصة الثانية.
- خوارزمية التعداد
- نريد أقل معدل page-fault

سنتحدث عن طريقة عمل كل خوارزمية وسيكون هنالك مثال لكل طريقة، وسنقيم كل الخوارزمية بحساب عدد فقدانها للصفحات page faults التي قد تحدث على سلسلة معينة طلبات الصفحات.

وتكون سلسلة طلبات الصفحات هي كتابة أرقام الصفحات حسب ترتيب طلبها، مثلا السلسلة التالية:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

توضح أننا نريد الصفحة رقم 1، ثم رقم 2، ثم رقم 3، وهكذا.

9.6.1. خوارزمية الصفحة الأقدم تحميلا (FIFO)

استبدال الصفحة الأقدم (أقدم صفحة تم تحميلها بالذاكرة). أي التعامل مع الصفحات الموجودة بالذاكرة حسب زمن وصولها، فالتي تصل إلى الذاكرة أولا تخرج من الذاكرة أولا. عيب هذه الخوارزمية أنه قد يكون لدينا صفحات في الذاكرة لغير مستخدمة لمدة طويلة.

مثال (6-9)

إذا كان لدي سلسلة طلبات الصفحات التالية:

1,2,3,2,1,5,2,1,6,2,5,6,3,1,3,6,1,2,4,3

ما هي عدد خطأ الصفحات (page fault) إذا استخدمنا 3 إطارات فارغة.

الحل

في البداية ستكون الإطارات الثلاثة فارغة، سنحتاج أولا للصفحة رقم 1 ولأنها غير موجودة (أول خطأ صفحة) سنحضرها في إطار من الثلاثة، بعدها سنحتاج الصفحة رقم 2 (غير موجودة مما سيسبب خطأ صفحة ثاني) سنحضرها في الإطار الثاني، بعدها سنحتاج إلى الصفحة رقم 3 (غير موجودة أيضا لذلك سنحضرها إلى الإطار الثالث). الآن عدد خطأ الصفحات هو 3، والإطارات الثلاثة ممتلئية.

سنحتاج بعدها الصفحة رقم 2 والصفحة رقم 1 (كلها موجودة بالذاكرة (لا يوجد خطأ صفحة)). بعدها سنحتاج الصفحة رقم 5 وهي غير موجودة بالذاكرة لذلك سيبب ذلك خطأ صفحة، هذا بالإضافة إلى أننا نريد استبدال صفحة لأن الإطارات الثلاث مليئة، لذلك سنخرج الصفحة الأقدم وهي الصفحة رقم 1 وندخل مكانها الصفحة رقم 5، الآن عدد خطأ الصفحات أصبح 4. سنحتاج الصفحة 2 وهي موجودة، بعدها سنحتاج الصفحة رقم 1، ولأنها غير موجودة فسيكون هنالك خطأ صفحة (أصبح عدد أخطاء الصفحات 6)، وسنستبدل الصفحة 2 مكان الصفحة 1 لأنها الأقدم. وهكذا ستتم عمليات الاستبدال إلى نهاية السلسلة كما في الشكل التالي بحصيلة عدد 14 خطأ صفحة..

1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
1	1	1			2		3	5	1	6		2	5		3		1	6	2
	2	2			3		5	1	6	2		5	3		1		6	2	4
		3			5		1	6	2	5		3	1		6		2	4	3

9.6.2. الخوارزمية المثلى

استبدال الصفحة التي لن نطلبها قريبا، أي الأبعد في السلسلة.

مثال (9-7)

إذا كان لدي سلسلة طلبات الصفحات التالية:

1,2,3,2,1,5,2,1,6,2,5,6,3,1,3,6,1,2,4,3

ما هي عدد أخطاء الصفحات (page fault) إذا استخدمنا 3 إطارات فارغة.

الحل

في البداية ستكون الإطارات الثلاثة فارغة، سنحضر الصفحات 1، 2، 3 في الإطارات الثلاث الفارغة مما يولد 3 خطأ صفحة. بعدها سنحتاج إلى الصفحات 2، 1 وهما بالذاكرة فنستخدمهما دون حدوث خطأ صفحة. بعدها سنحتاج الصفحة 5، وهي ليست بالذاكرة لذلك لابد من إخراج صفحة وإدخالها مكانها. سنخرج الصفحة التي لا نطلبها قريبا (لدينا الآن بالذاكرة الصفحات 1،2، 3) فإيها سنخرج:

- الصفحة 1 سنطلبها بعد صفحتين من الصفحة 5ز
- الصفحة 2 سنطلبها بعد الصفحة 5 مباشرة.
- الصفحة 3 سنطلبها بعد 6 صفحات، وهي الأبعد لذلك سنخرجها ونضع 5 مكانها، الآن لدينا بالذاكرة الصفحات 1،2، 5.

سنحتاج الصفحة 2 وهي موجودة بالذاكرة، وبعدها سنحتاج الصفحة 1 وهي بالذاكرة، بعدها سنحتاج الصفحة 6 التي لا توجد بالذاكرة لذلك سنتبدلها بالصفحة 1 لأنها الأبعد. وهكذا إلى أن نصل للحل التالي:

1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
1	1	1			1			6				6	6				2	2	
	2	2			2			2				2	1				1	4	
		3			5			5				3	3				3	3	

سيكون لدينا في النهاية 9 خطأ صفحة.

9.6.3. التي لم تستخدم حديثا ((least recently used (LRU)

بإفتراض أن الصفحة التي استخدمت حديثا هي التي سنحتاجها وهي التي ستستخدم مرة ثانية، أما التي لم تستخدم حديثا فغالبا لن نحتاجها في القريب العاجل. لذلك سنخرج الصفحة التي لم تستخدم حديثا (الأقدم استخداما).

مثال (9-8)

إذا كان لدي سلسلة طلبات الصفحات التالية:

1,2,3,2,1,5,2,1,6,2,5,6,3,1,3,6,1,2,4,3

ما هي عدد أخطاء الصفحات (page fault) إذا استخدمنا 3 إطارات فارغة.

الحل

في البداية ستكون الإطارات الثلاثة فارغة، سنحضر الصفحات 1، 2، 3 في الإطارات الثلاث الفارغة مما يولد 3 خطأ صفحة. بعدها سنحتاج إلى الصفحات 2، 1 وهما بالذاكرة فنستخدمهما دون حدوث خطأ صفحة. بعدها سنحتاج الصفحة 5، وهي ليست بالذاكرة لذلك لابد من إخراج صفحة وإدخالها مكانها. سنخرج الصفحة الأقل استخداما (لدينا الآن بالذاكرة الصفحات 1،2، 3) فإيها سنخرج:

- الصفحة 1 استخدمت آخر شئ.

- الصفحة 2 استخدمت قبل الصفحة 1.
- الصفحة 3 استخدمت قبل الصفحة 2،أي هي الأول استخداما بين الصفحات الثلاث التي بالذاكرة، لذلك سنخرجها ونضع 5 مكانها، الآن لدينا بالذاكرة الصفحات 1،2، 5.

سنحتاج الصفحة 2 وهي موجودة بالذاكرة، وبعدها سنحتاج الصفحة 1 وهي بالذاكرة، بعدها سنحتاج الصفحة 6 وهي ليست بالذاكرة لذلك سنتبدلها بالصفحة 5 لأنها الأقل استخداما. وهكذا إلى أن نصل للحل التالي:

1	2	3	2	1	5	2	1	6	2	5	6	3	1	3	6	1	2	4	3
1	1	1			2			2		6		5	6				6	1	2
	2	2			1			1		2		6	3				1	2	4
		3			5			6		5		3	1				2	4	3

سيكون لدينا في النهاية 11 خطأ صفحة.

9.6.4. الخوارزميات المعتمدة على العدد (counting-based)

تعتمد هذه الخوارزميات على وجود عدادات لحساب عدد مرات استخدام الصفحات. حيث يكون هنالك عداد لكل صفحة وكلما استخدمت صفحة يزيد عدادها بواحد. منها:

9.6.4.1. خوارزمية الصفحة الأقل استخداما ((least frequently used (LFU)

بإفتراض أن الصفحات الاكثر استخدامها (قيمة عداداها أكبر) هي النشطة وهي التي ستستخدم كثيرا، فنقوم بإخراج الصفحة التي لها أقل رقم في عدادها، أي عدد مرات استخدامها أقل من بقية الصفحات.

لكن هذه الخوارزمية قد يكون بها مشكلة إذا كانت الصفحة ذات العداد الأكبر لن تعمل مرة أخرى، فتظل في الذاكرة دون فائدة منها. ويمكن حل مثل هذه المشكلة بنقص عدادها بفترات منتظمة.

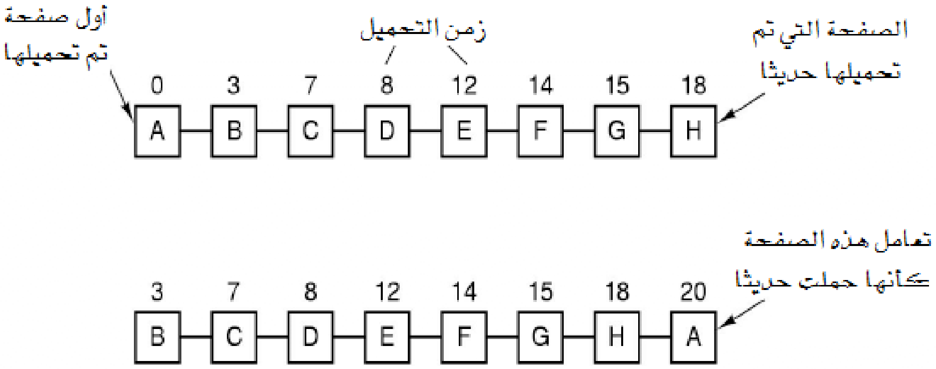
9.6.4.2. خوارزمية الصفحة الأكثر استخداما ((most frequently use (MFU)

هنا نفترض أن الصفحات التي لها قيمة عداد أقل هي التي وصلت حديثا وهي التي تحتاج إلى وقت اكثر في الذاكرة، لذلك سنخرج الصفحة التي لها رقم أكبر في عدادها (التي استخدمت كثيرا).

تعتبر هذه الخوارزميات غير شائعة لأن تطبيقها مكلف.

9.6.5. خوارزمية الفرصة الثانية (Second chance)

تعتبر هذه الخوارزمية تعديل لخوارزمية الأقدم تحميلا أولا (FIFO)، الفرق أننا سنختبر الصفحة الأقدم فإذا كانت قد استخدمت حديثا فإننا سنضعها في نهاية الصف ونعتبرها وصلت حديثا ونختبر الصفحة التي تليها، أما إذا لم تستخدم حديثا فسنخرجها. أنظر الشكل التالي.



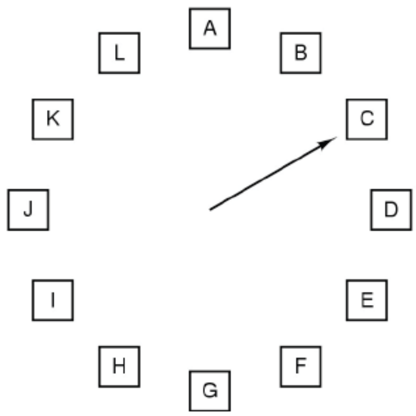
تنبيه:

نتعرف على الصفحة هل استخدمت أم لا بإختبار خانة الاستخدام التي توجد في كل صفحة فإذا كانت هذه الخانة تحتوى على صفر (clear) فهذا يعني أنه لم تستخدم حاليا، أما إذا كانت تحتوي قيمة (set) فيعني هذا أنه تم استخدامها حديثا. عند وجود خانة الاستخدام تحتوي قيمة سنقوم تفريغها من القيمة (clearit) قبل أن نرجعها إلى نهاية الصف.

9.6.6. خوارزمية الساعة (Clock)

أشبه في طريقة عملها بخوارزمية الفرصة الثانية لكنها أكثر كفاءة في التطبيق. سيتم إختبار الصفحة التي يشير إليها مؤشر الساعة هل استخدمت حديثا أم لا، فإذا استخدمت حديثا نحول مؤشر الساعة ليشير إلى الصفحة التي تليها ونختبرها هل استخدمت حديثا أم لا، فإذا استخدمت ننقل المؤشر للصفحة التي تليها، وهكذا حتى نعثير على صفحة لم تستخدم فنخرجها.

في الشكل التالي سنختبر الصفحة C فإذا استخدمت سيتحول المؤشر إلى الصفحة D ونختبرها هل استخدمت أم لا، وهكذا حتى نعثر على صفحة لم تستخدم فنستبدلها ويتحول المؤشر إلى الصفحة التي تليها.



9.6.7. برنامج محاكاة خوارزميات إستبدال الصفحات

يمكنك كتابة برنامج بأي لغة (C/C++ أو جافا) مثلا، لمحكاة عمل خوارزميات استبدال الصفحات، حيث يطلب البرنامج من المستخدم إدخال نوع الخوارزمية وعدد الإطارات الفارغة وسلسلة الصفحات فيحسب خطوات الاستبدال وعدد أخطاء الصفحات . مثلا:

```
Choose page replacement algorithm:
   1. FIFO
   2. Optimal
   3. LRU
1
Enter number of pages in logical memory:
4
Enter number of frames in physical memory:
2
```

```
Enter reference string:
0 1 2 1 3 1 3 2 1 3 1
Solution:
0: PAGE FAULT -- added at frame 0
1: PAGE FAULT -- added at frame 1
2: PAGE FAULT -- replaces page 0 at frame 0
1: found at frame 1
3: PAGE FAULT -- replaces page 1 at frame 1
1: PAGE FAULT -- replaces page 2 at frame 0
3: found at frame 1
2: PAGE FAULT -- replaces page 3 at frame 1
1: found at frame 0
3: PAGE FAULT -- replaces page 1 at frame 0
1: PAGE FAULT -- replaces page 2 at frame 1
Number of page fault= 8
```

طبق البرنامج على السلسلة التالية (في 8 إطارات فارغة):

```
0 1 2 3 4 8 9 10 11 12 5 6 7 8 2 3 4 8 9 10 11 0 1 2
5 6 5 6 5 6 13 14 15 0 1 2 3 4 8 9 10 11 12 13 14 15
```

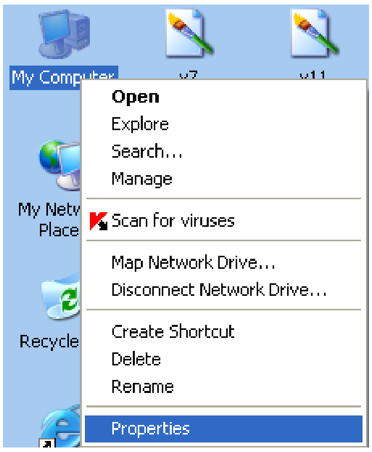
9.7. عملي (التحكم في الذاكرة الظاهرية)

نظم التشغيل مثل ويندوز تحجز جزء من القرص الصلب ليستخدم كذاكرة ظاهرية، ويتيح لنا نظام التشغيل إمكانية زيادة أو تقليل هذه المساحة المحجوزة للذاكرة الظاهرية.

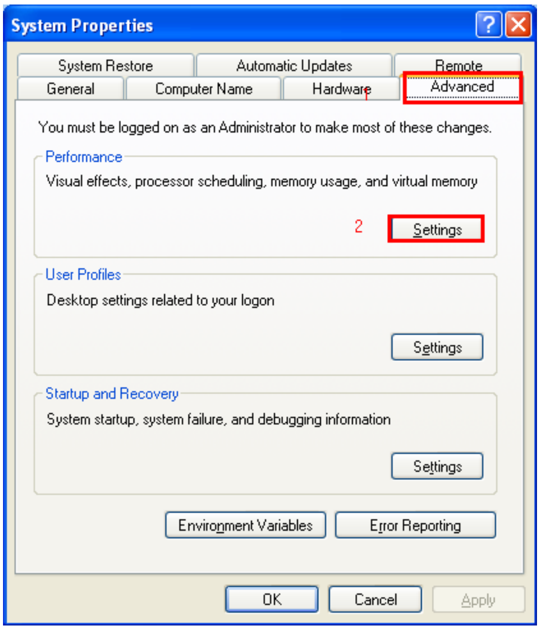
ولكن لماذا نغير حجم الذاكرة الظاهرية؟ لأن هنالك برامج كبيرة لا تكفي الذاكرة الظاهرية الحالية لتخزينها، أو قد تظهر رسالة تخبرك بأن الذاكرة الظاهرية غير كافية. ظهور مثل هذه الرسالة هو بسبب أنك أردت تشغيل برنامج كبير الحجم ولا تكفى الذاكرة الظاهرية لتشغيله، لذلك عليك زيادة حجم الذاكرة الظاهرية بجهازك.

الخطوات التالية توضح كيف يمكنني تغيير مساحة الذاكرة الظاهرية في ويندوز XP:

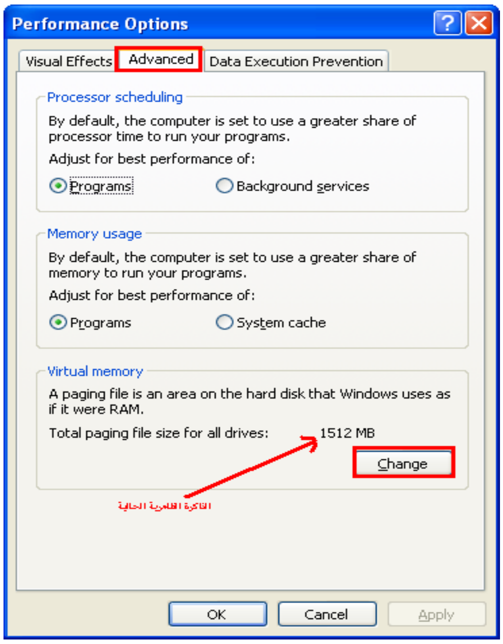
أنقر يمين الماوس على My Computer ثم نختار Properties.

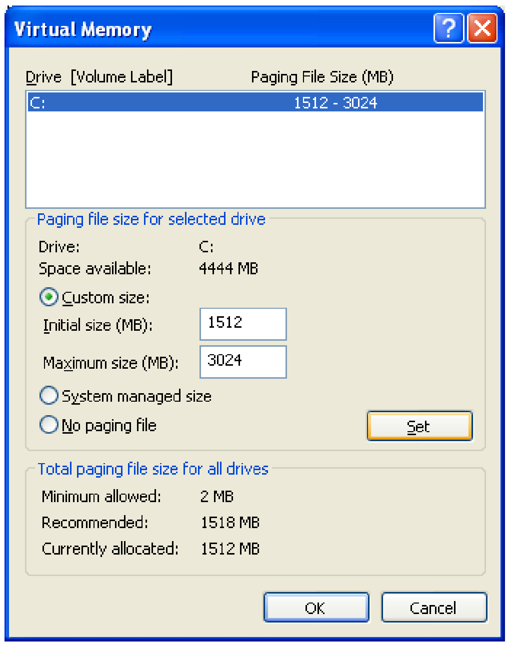


أنقر على التبويب Advanced ثم على Settings في Performance

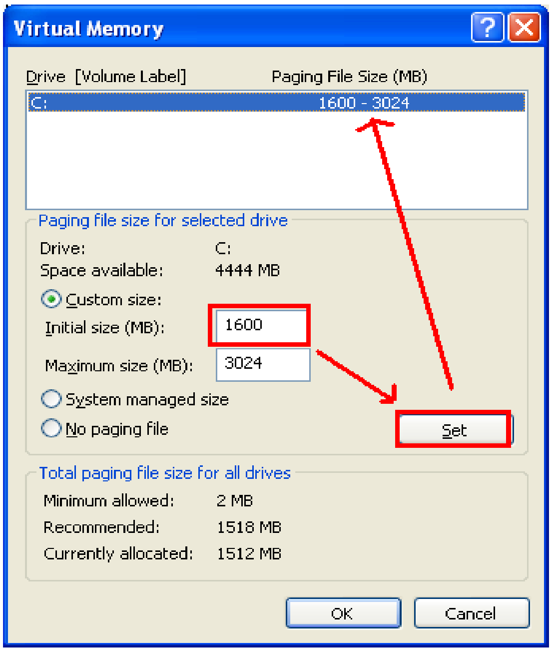


من النافذة التالية اختار التبويب Advanced ثم Change





عدل القيمة أمام Initial size بالميغابايت ثم انقر على Set


Virtual Memory
Drive [Volume Label]
Paging File Size (MB)
C:
1600 - 3024
Paging file size for selected drive
Drive:
C:
Space available:
4444 MB
Custom size:
Initial size (MB):
1600
Maximum size (MB):
3024
System managed size
No paging file
Set
Total paging file size for all drives
Minimum allowed:
2 MB
Recommended:
1518 MB
Currently allocated:
1512 MB
OK
Cancel

9.8. تمارين محلولة

1. إذا كان لدينا سلسلة الصفحات التالية (page refrence string):

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6.

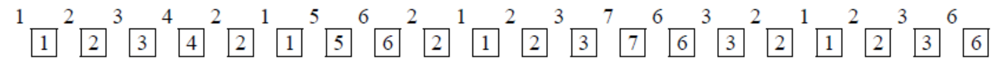
أحسب عدد أخطاء الصفحات (page faults) للخوارزميات التالية بإفتراض إطار واحد، إطارين، ثلاث إطارات، أربع إطارات، خمسة إطارات، ستة إطارات، وسبعة إطارات:

* خوارزمية FIFO

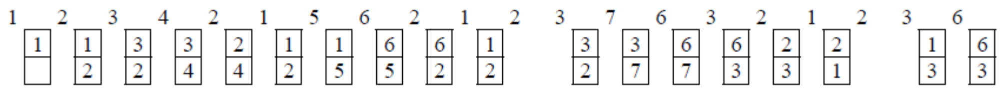
* خوارزمية LRU

الحل

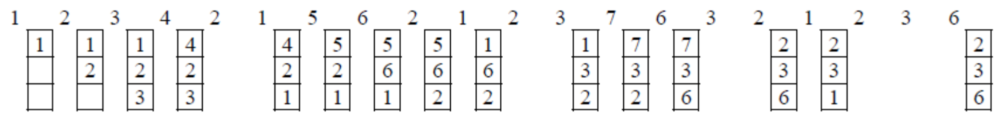
لخوارزمية LRU



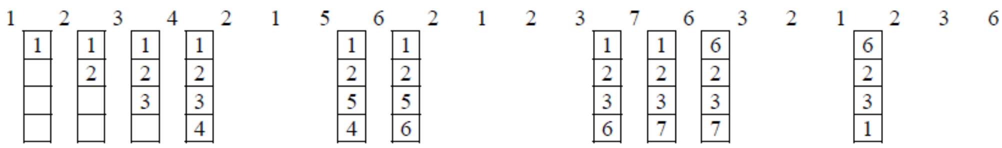
20 خطأ صفحة



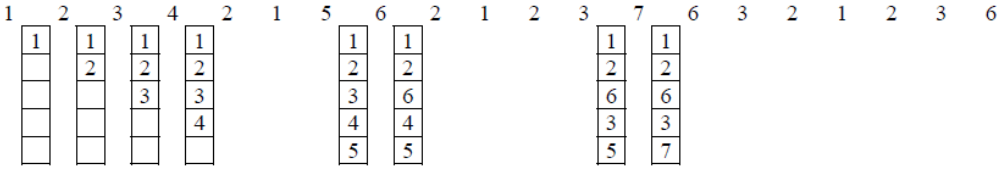
18 خطأ صفحة



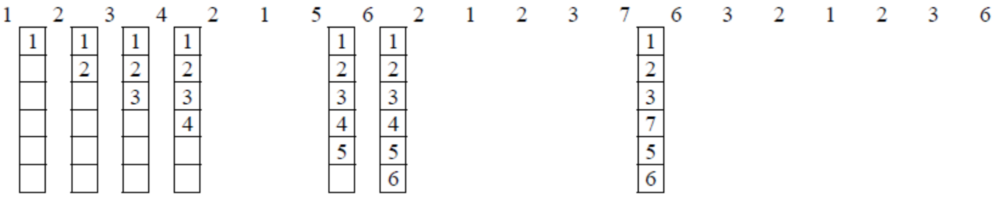
15 خطأ صفحة



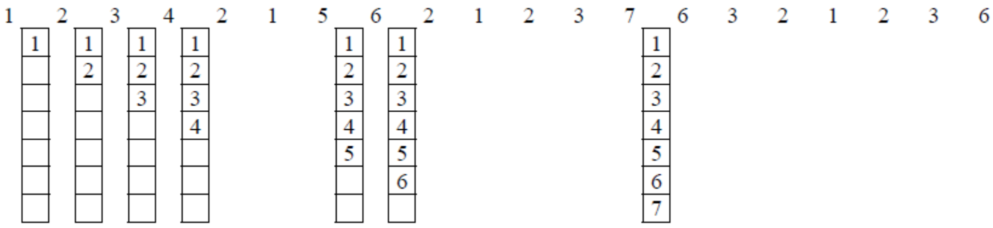
10 خطأ صفحة



8 خطأ صفحة.

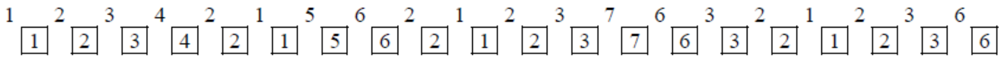


7 خطأ صفحة.

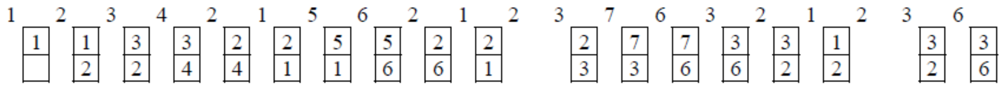


7 خطأ صفحة

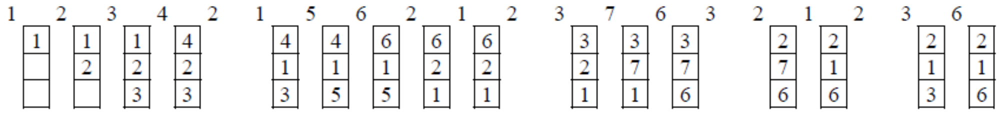
لخوارزمية FIFO:



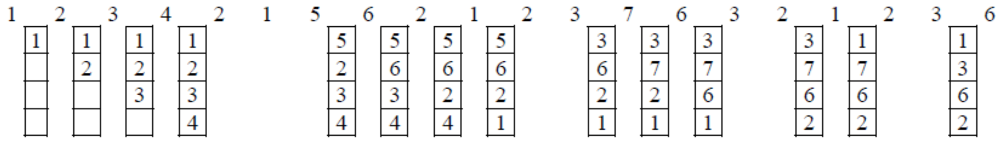
20 خطأ صفحة.



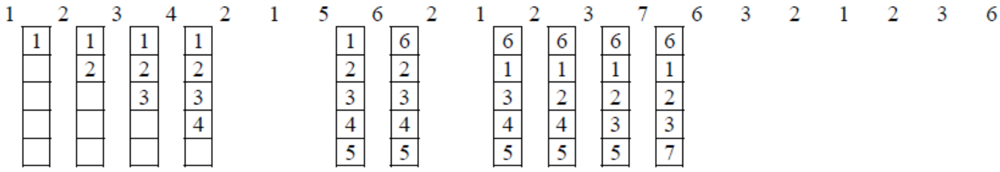
18 خطأ صفحة.



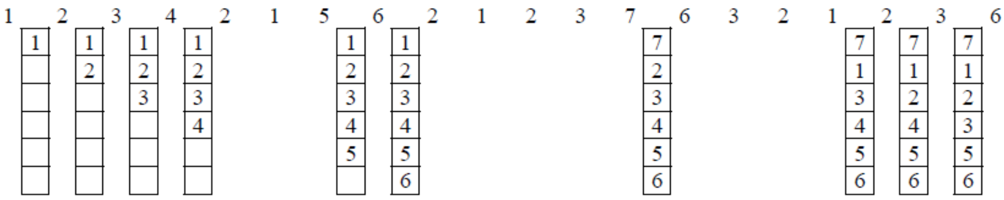
16 خطأ صفحة.



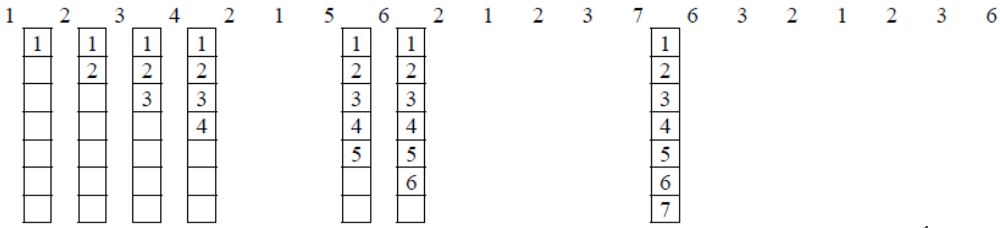
14 خطأ صفحة.



10 خطأ صفحة.



10 خطأ صفحة.



7 خطأ صفحة.

9.9. تمارين غير محلولة

2. عرف الذاكرة الظاهرية ؟
3. تكون البرامج التي تستخدم الذاكرة الظاهرية بطيئة نوع ما، لماذا ؟
4. ما هي المبادلة (swapping) ؟
5. تنقسم المبادلة إلى نوعين ، ما هما ؟
6. اذكر ثلاث من خوارزميات تبديل الصفحات ؟
7. إذا كان لدي سلسلة طلبات الصفحات التالية:

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

- أحسب عدد أخطاء الصفحات (page fault) إذا استخدمنا خوارزمية FIFO لعدد:
  - إطار واحد فارغ.
  - إطارين فارغين .
  - ثلاث إطارات فارغة.
  - أربع إطارات فارغة.
  - خمسة إطارات فارغة.
  - ستة إطارات فارغة.
  - سبعة إطارات فارغة.
- كم ستتوقع عدد أخطاء الصفحات لثمانية إطارات فارغة ؟
- ما هي العلاقة التي يمكن استنتاجها بين عدد الإطارات الفارغة وعدد خطأ الصفحات ؟

8. أحسب عدد خطأ الصفحات للسلسلة التالية:

7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

إذا كان لدينا ثلاث إطارات فارغة، باستخدام:

- خوارزمية FIFO ؟
- الخوارزمية المثلى (optimal) ؟
- خوارزمية LRU ؟
- أي خوارزمية هي الأفضل (أقل عدد أخطاء صفحات) ؟ ولماذا ؟

9. أختار الإجابة الصحيحة (قد يكون لدينا أكثر من إجابة صحيحة)

- إذا كان لدينا السلسلة التالية من الصفحات :

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

- فإن عدد خطأ الصفحات (في أربعة إطارات) باستخدام الخوارزمية FIFO:
  - 9
  - 12
  - 14
  - 20
  - لا شي مما ذكر صحيح والعدد هو _______________

- تبديل العملية خارج الذاكرة يسمى :
  - SWAP IN
  - SWAP OUT
  - SWAPPING
  - STORING

- إذا كان لدينا السلسلة التالية من الصفحات :

1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6

فإن عدد خطأ الصفحات (في سبعة إطارات) باستخدام الخوارزمية FIFO هو:

- 9
- 12
- 14
- 20
- لا شي مما ذكر صحيح والعدد هو ______________

- عند ما يحدث خطأ صفحة فإن أول خطوة يقوم بها نظام التشغيل هي:
  - الحصول على إطار فارغ.
  - تبديل الصفحة المطلوبة في الفريم.
  - تحديث الجداول.
  - التأكد من أن ليست الصفحة ليست صحيحة.
  - التأكد من أن الصفحة ليست بالذاكرة.
  - تعديل بت التصحيح إلى v.
  - إعادة تنفيذ الأمر الذي سبب الـ page fault

الباب العاشر:

مدير الأجهزة

الباب العاشر

مدير الأجهزة (Devices Manager)

10.1. مقدمة

معظم أجهزة التي ترتبط بالحاسب هي بغرض إدخال بيانات لوحدة المعالجة (التي تتكون من الذاكرة والمعالج) أو لإظهار نتائج من هذه الوحدة. التعامل مع هذه الأجهزة مباشرة يعتبر عملية معقدة وصعبة وتتطلب إلمام بتفاصيل عمل الجهاز وكيف يتم برمجته (بلغة الآلة) لينفذ عمل ما. الحمد لله على نعمة نظام التشغيل، فقد أغنانا عن معرفة هذه التفاصيل الدقيقة وتعلم لغة الآلة وبرمجة كل جهاز على حدة. فقد قام هو بذلك نيابة عنا وتولى التعامل مع هذه التفاصيل. فهنالك جزء بنظام التشغيل يسمى مدير الأجهزة قد صمم من أجل التحكم في الأجهزة وإدارتها. فهو يرسل الأوامر للأجهزة، يتلقى معلومات منها، يكتشف أخطاءها ويعالجها.

يخفي نظام التشغيل تفاصيل أجهزة الدخل والخرج ويوفر واجهة موحدة للتعامل معها، يتم ذلك عبر برمجيات تسمى برمجيات الدخل والخرج .

10.2. أجهزة الدخل والخرج

تختلف الأجهزة في مهامها وسعة تخزينها وسرعاتها، وتقاس سرعة الجهاز بكمية البيانات التي يمكنه التعامل معها في فترة زمنية معينة. فمثلا لوحة المفاتيح تقاس سرعتها بكمية البايتات التي يتعامل معها في الثانية.

> نلاحظ أننا الآن نتحدث عن مقاسات بالثواني، بينما كان حديثا في المعالج والذاكرة عن مقاسات بالنانوثانية.

نظام التشغيل مسئول من التعامل مع العديد من الأجهزة الطرفية (peripheral devices) منها الأقراص وفيها لوحة المفاتيح ، الماوس، الشاشة ، الطابعات ، كرت الشبكة ، المودم ، موانئ الدخل والخرج.

يمكن أن نطلق كلمة جهاز (device) على أي قطعة إلكترونية أو إلكتروميكانيكية من المكونات المادية تساهم في إدخال أو إخراج معلومة للحاسب.

يمكن تقسيم الأجهزة إلى نوعين:

- أجهزة تعمل بنظام الكتل (block).
- أجهزة تعمل بنظام الحروف (character).

النوع الأول يرسل المعلومات في شكل كتل ذات حجم ثابت ولكل كتلة عنوان، مثال لهذا النوع القرص الصلب. النوع الآخر يرسل المعلومات في شكل سلسلة من الحروف، ليس لها عنوان ولا نجري عليها عمليات بحث، مثل الطابعة، كرت الشبكة، ولوحة المفاتيح.

هنالك بعض الأجهزة لا تنتمي إلى أي نوع من النوعين أعلاه مثل الساعة، التي فقط تقوم بإنشاء المقاطعات (interrupts).

يقوم مدير الأجهزة بنظام التشغيل بالتعامل مع هذه الأجهزة مخفياً عنا تفاصيلها المزعجة وموفرا لنا طريقة سهلة للتعامل معها.

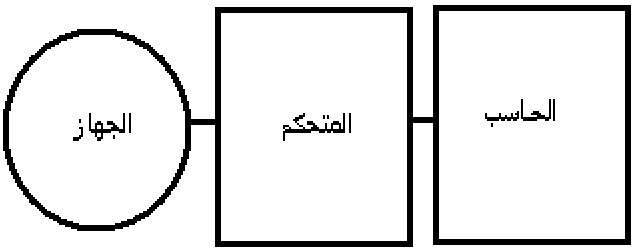
بعض نظم التشغيل مثل ينكس تقوم بإضافة برامج خاصة لإدارة والتعامل مع هذه الأجهزة يسمى سواقات الأجهزة (device drivers) أو كما نطلق عليها نحن التعريفات. فمثلا عندما تشتري كرت مودم أو طابعة مثلا فإنك تجد برنامج تعريف مرفق مع الجهاز، هذا البرنامج يثبت على نظام التشغيل ليتمكن الأخير من تشغيله والتعامل معه.

التعريف (device driver) هو برنامج يتم كتابته للجهاز الجديد (new device) بواسطة الشركة المصنعة للجهاز، ثم يضاف إلى نظام التشغيل بحيث يستخدم نظام التشغيل هذا التعريف للتعامل مع الجهاز الجديد.

10.3. المتحكم (*controller*)

يتكون كل جهاز من شقين:

1. المتحكم (controller) أو المحول (adapter).
2. الجهاز نفسه.



شكل رقم (10-1): المتحكم بين الجهاز والحاسب.

المتحكم يعتبر وسيط بين الحاسب والجهاز، فهو الذي يعرف التفاصيل الدقيقة لطريقة عمل الجهاز وهو الذي يتعامل مع الجهاز مباشرة. أما الحاسب فيتعامل الجهاز بصورة غير مباشرة وذلك عبر المتحكم.

نحن وبرامجنا نتعامل مع نظام التشغيل حيث نطلب منه ما نريد، فيقوم هو بدوره بالتعامل مع المتحكم (بوضع قيم معينة في مسجلات تحكم معينة)، فيقوم المتحكم بالاتصال بالجهاز، فينفذ الجهاز العمل المطلوب.

القيم التي يضعها مدير الأجهزة في مسجلات التحكم ترسل أوامره للجهاز والتي قد تكون:

- إرسال أو استلام بيانات.
- فتح أو غلق الجهاز (On or off).
- تنفيذ أي مهمة أخرى.

بالإضافة إلى هذه المسجلات، يوجد لدى بعض الأجهزة خازن بيانات (buffer)، يستخدمه مدير الأجهزة للقراءة أو الكتابة من الجهاز.

مثال (10-1)

يوجد خازن في كرت الشاشة يسمى ذاكرة الفيديو (video ram)، لعرض أي معلومات على الشاشة، يقوم مدير الأجهزة بكتابة ما يريد عرضه في هذه الذاكرة، ويضع في مسجلات التحكم أمر إظهار محتوي ذاكرة الفيديو على الشاشة.

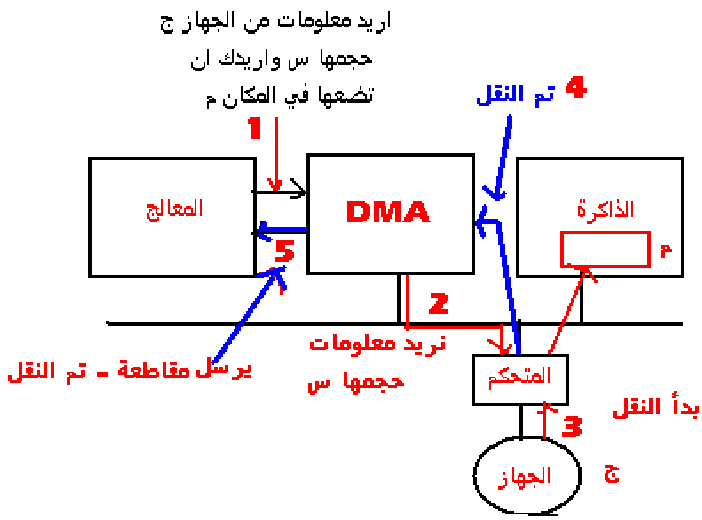
10.4. الوصول المباشر للذاكرة (***Direct Memory Access (DMA)***)

في الحاسبات الحديثة توجد قطعة إلكترونية تسمى DMA توجد غالبا باللوحة الأم، تتعامل مع الذاكرة باستغلالية عن المعالج. هنالك العديد من المتحكمات (controllers) تستخدم DMA مثل كرت الصوت، كرت الشاشة، ومتحكم القرص الصلب. تستطيع DMA نقل البيانات بين الأجهزة والذاكرة دون إشغال المعالج بذلك.

> في الحواسيب القديمة والتي لا يوجد بها DMA يكون المعالج في حالة متابعة لعمليات الدخل والخرج، حيث يتابع عملية نقل البيانات من الجهاز إلى الذاكرة. في هذه الأثناء لن يستطيع المعالج تنفيذ أي أوامر أخرى مما يقلل أداءه. فعمليات الدخل والخرج تأخذ الكثير من زمن المعالج.

يعمل DMA كمدير تنفيذي للمعالج، حيث يقوم بعمليات الدخل والخرج نيابة عن المعالج. فإذا أراد المعالج معلومات من جهاز يقوم بالآتي:

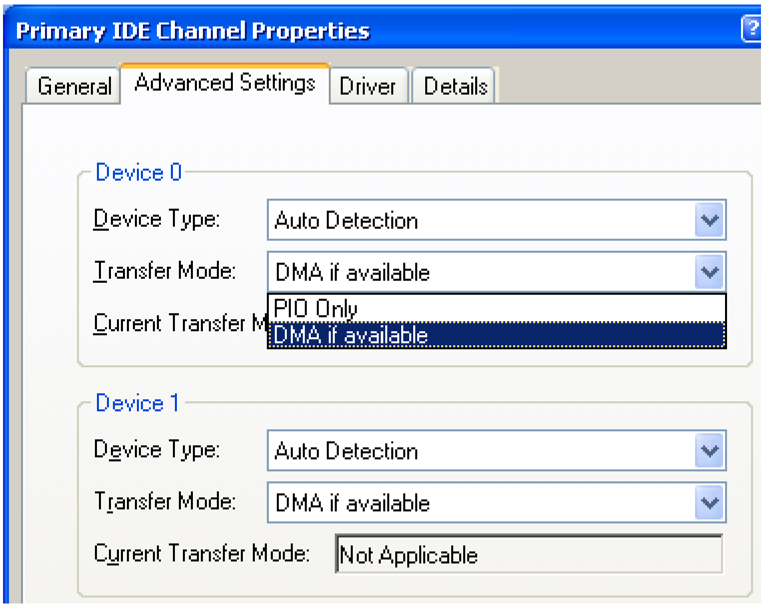
- يهيئ DMA لعمليات نقل البيانات (حجم البيانات المراد نقلها، الجهاز المنقولة منه البيانات، المكان الذي سيتم تخزينها فيه بالذاكرة (عنوان الذاكرة)).
- ثم يذهب المعالج لأداء أعمال أخرى (مستفيدا من وقته).
- في هذه الأثناء يقوم DMA بعمليات نقل البيانات.
- عند ما يفرغ DMA من عمله يرسل إشارة مقاطعة (interrupt) للمعالج يخبره فيها بأن عملية نقل البيانات قد تمت.



شكل رقم (10-2): كيف يعمل DMA.

10.4.1. تشغيل أو تعطيل DMA في ويندوز XP

يمكنك تعطيل أو تفعيل DMA بجهازك وذلك بفتح مدير الأجهزة (device manager)، ثم النقر على IDE ATA/ATAPI controllers لتتوسع فتظهر إيقونات فرعية هي للأقراص الصلبة وسواقات الاسطوانات. أنقر على أي إيقونة أمامها كلمة channel بيمين الماوس ثم اختار خصائص (properties)، فتظهر شاشة، أختار منها التبويب Advanced settings، ثم أختار من أمام Transfer Mode للجهاز الذي تريد، حالة نمط النقل، الشكل (10-3-). تفعيل DMA يجعل القرص الصلب والاسطوانات الضوئية من نقل البيانات للذاكرة مباشرة (دون مرورها بالمعالج)، مما يقلل الجهد على المعالج.



شكل رقم (3-10): تفعيل DMA أو تعطيلها.

10.5. أهداف مدير الأجهزة

10.5.1. الكفاءة

من المعلوم أن الأجهزة تعتبر بطيئة إذا ما قورنت بسرعة المعالج وسرعة الذاكرة. وبالتالي فإن كفاءة الحاسب ككل وسرعته مرتبطة بسرعة أبطأ جهاز فيه، فمثلا قد ينقل القرص البيانات بسرعة 1 ميغابايت في الثانية بينما تنقلها لوحة المفاتيح بسرعة 3 بايت في الثانية. مهمة مدير الأجهزة هنا أن يحسن الكفاءة وذلك بتقليل زمن انتظار المعالج لأجهزة الدخل والخرج بقدر المستطاع. الخازن الموجود بالمتحكم وجهاز DMA تم إضافتهم للمساهمة في هذا التحسين.

10.5.2. استغلالية الأجهزة

يجب على البرامج أن تعمل باستغلالية عن الأجهزة التي تتعامل معها، فلا يجدر بي أن أعرف نوع الطابعة مثلا أو الشركة المصنعة لها لأتعامل معها. وإذا تم تغير الطابعة بطابعة أخرى فعلى البرامج أن لا تغير من شفراتها وأن تعمل بنفس الطريقة السابقة. ولكي نصل لهذا الهدف لابد لعمل واجهة برمجية للأجهزة يتعامل

معها المستخدم وبرامجه تكون بعيدة كل البعد عن الأجهزة ويقوم نظام التشغيل بالتعامل مع الأجهزة الحقيقية بينما تمثل الواجهة التي يتعامل معها المستخدم أجهزة افتراضية تستقبل طلباتنا وتمررها لنظام التشغيل.

10.5.3. المشاركة

الكثير من الأجهزة نحتاج أن تكون مشتركة بين المستخدمين والبرامج، فعلى نظام التشغيل أن يوفر هذه الأجهزة بين المستخدمين بصورة عادلة، بحيث لا يحتكر أحد البرامج الجهاز المشترك دون بقية البرامج.

10.5.4. الحماية

كل الأجهزة يجب حمايتها من الاستخدام غير الرشيد، فالأجهزة المخصصة يجب حمايتها من استخدامها بأكثر من عملية في نفس الوقت مثلا.

10.5.5. معالجة الأخطاء

إذا حدث خطأ في جهاز، كأن نحاول الكتابة على جهاز دخل أو القراءة من جهاز خرج، فعلى مدير الأجهزة اكتشاف الخطأ ومعالجته إذا كان بالإمكان ذلك أو إرسال تقرير للبرنامج الذي سبب الخطأ.

10.6. قواعد برمجيات الدخل والخرج
Principles of I/O Software

القواعد التي نراعيها في تصميم برمجيات الدخل والخرج تسعى لتحقيق أهداف مدير الأجهزة ، وهي كالتالي:

- إستقلالية الاجهزة (device independence)، كتابة برامج تتعامل مع الأجهزة دون معرفة تفاصيل الأجهزة مسبقا. مثلا يمكن استدعاء نفس نداء النظام لقراءة ملف من القرص الصلب أو القرص المرن أو الأسطوانة الضوئية.

- توحيد التسمية (uniform naming): اسم الملف أو الجهاز لا يعتمد على خصائص القرص. مثلا في ينكس كل شئ يتعبر ملف.

- معالجة الأخطاء (error handling): معالجة الأخطاء في أدنى مستوى لها. مثلا إذا تم إكتشاف الخطأ فيحاول المتحكم معالجة الخطأ، وإلا فعلى التعريف (device driver) محاولة تصحيح الخطأ، ويمرر الخطأ للطبقات الأعلى إذا فشل التعريف في معالج الخطأ.

- النقل المتزامن والغير متزامن (*Synchronous* vs. *asynchronous* transfers): معظم عمليات الدخل والخرج غير متزامنة، حيث يتم برمجة DMA ليقوم بالعمل ثم القيام بأعمال اخرى وعندما يكتمل العمل ترسل مقاطعة. برامج المستخدم عادة لا تحتاج أن تري أو تتعامل مع المقاطعات. وقد توفر التعريفات (device driver) نداءات نظام تقوم بالحجز (block) عند الإنتظار وفك الحجز (unblock) عند وصول المقاطعات.

- التخزين المؤقت (buffering): البيانات التي ترد من الجهاز لا تخزن في وجهتها النهائية في البداية وإنما يجب فحصها قبل إرسالها إلى وجهتها النهائية، لذلك تخزن مؤقتا في خازن قد يوجد في متحكم الجهاز.

- الأجهزة المشتركة (shared devices) والمخصصة (dedicated): هنالك بعض الأجهزة التي تكون مشتركة بين أكثر من مستخدم، فمثلا يمكن أن يكون هنالك عدد من المستخدمين يفتحون العديد من الملفات الموجودة بالقرص الصلب. وهنالك أجهزة لا تقبل المشاركة مثل كتابة كتل في الشريط الممغنط (tape). على نظام التشغيل دعم التعامل مع هذه الأنواع، فيوفر طرق مختلفة للتعامل مع الأنواع المختلفة.

10.7. طرق الدخل والخرج

يتم الدخل والخرج بأساليب مختلفة منها:

- الدخل والخرج المبرمج.
- الدخل والخرج بالمقاطعات.

- o الدخل والخرج باستخدام DMA.

10.7.1. الدخل والخرج المبرمج (*Programmed I/O*)

هنا يتم التعامل مع الجهاز بالخطوات التالية:

1. اختبار الجهاز هل هو جاهز أم لا
2. محاولة استقبال/إرسال جزء من البيانات من/إلى الجهاز
3. تم إرسال جزء من البيانات.
4. إذا لم يكتمل الإرسال/الإستقبال ، إذهب للخطوة 1.

هنا مثلا إذا أردنا طباعة الحروف OSMAN في الطابعة، تقوم العملية بالتالي:

- طلب الاستحواذ على الطابعة.
- إذا كانت الطابعة غير متاحة ننتظر
- الطابعة متاحة أرسل الحرف الأول إلى ذاكرة الطابعة
- إنتظار أن تنهي الطابعة من طباعة هذا الحرف.
- اختبار الطابعة مرة أخرى وإرسال الحرف الثاني وهكذا.

هنا سيكون المعالج متابعا لإرسال جيمع الحروف وينتظر طباعتها واحد تلو الآخرن مما يجعله مشغولا حتى يتم طباعة السلسلة.

خلاصة القول أنه لن يستطيع المعالج القيام بعمل آخر قبل أن ينتهي من هذا العمل.

10.7.2. الدخل والخرج بالمقاطعات (*Interrupt-Driven I/O*)

في الطريقة أعلاه ينتظر المعالج الطابعة حتى تفرغ من طباعة الحرف الأول ثم ترسل له الحرف الثاني وهكذا.

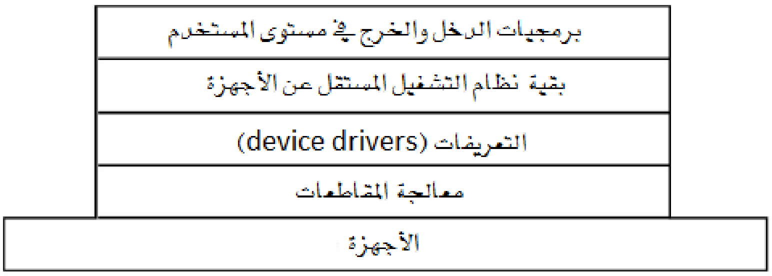
أما هنا فيقوم المعالج بإرسال الحرف الأول المراد طباعته إلى الطابعة ثم القيام بأي عمل آخر، وعندما تنتهي الطابعة منه طباعتها ترسل مقاطعة للمعالج ليرسل الحرف التالي وهكذا لن يضطر المعالج لإنتظار الطابعة وإنما هي تخبره متى ما كانت جاهزة.

10.7.3. الدخل والخرج بواسطة DMA (*I/O Using DMA*)

في الدخل والخرج بالمقاطعات فإن المقاطعة تحدث عند إكتمال طباعة كل حرف، وهذا يتسبب في هدر زمن المعالج. عدد المقاطعات ستكون بعدد الحروف المراد طباعتها. استخدام DMA يعني أن متحكم DMA يتابع طباعة الحروف في الطابعة واحد تلو الآخر دون إزعاج المعالج وهذا يقلل عدد المقاطعات من مقاطعة لكل حرف إلى مقاطعة واحدة عند طباعة كل الحروف.

10.8. طبقات برمجيات الدخل والخرج (***I/O software layers***)

تتكون برمجيات الدخل والخرج من طبقات، الطبقات الدنيا تهتم بخصوصيات أجهزة الدخل والخرج بينما توفر الطبقات العليا واجهة موحدة للمستخدم. هنالك أربع طبقات في برمجيات الدخل والخرج، أنظر الشكل (10-4)، حيث تقوم كل طبقة بوظيفة.



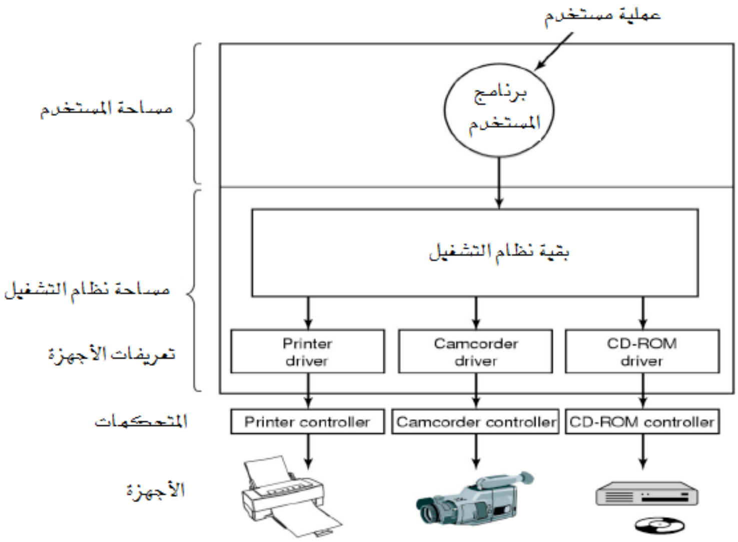
شكل رقم (10-4): طبقات برمجيات الدخل والخرج.

10.8.1. معالجة المقاطعات

من الأفضل إخفاء المقاطعات، وأفضل طريقة لذلك هي جعل برنامج التعريف (driver) الذي طلب عملية الدخل أو الخرج التوقف ريثما ينتهي الدخل أو الخرج وحدوث المقاطعة. بعدها يقوم إجراء المقاطعة بمهمته، ثم يواصل برنامج التعريف.

10.8.2. التعريفات (device drivers)

عادة تكتب الشركة المصنعة للجهاز التعريفات الخاصة بهذا الجهاز. وعند شرائك لأي جهاز تجد أسطوانة تعريف مرفقة معه. نقوم عادة بتثبيت التعريف في نظام التشغيل مما يجعله جزءا من نظام التشغيل. وبالتالي يستطيع نظام التشغيل من التعامل مع الجهاز. تعمل هذه التعريفات لإدارة الجهاز عبر متحكم الجهاز. ويتعامل نظام التشغيل مع الجهاز عبر تعريف الجهاز، الشكل (10-5).



شكل رقم (10-5): الأجهزة والتعريفات وربطها من نظام التشغيل.

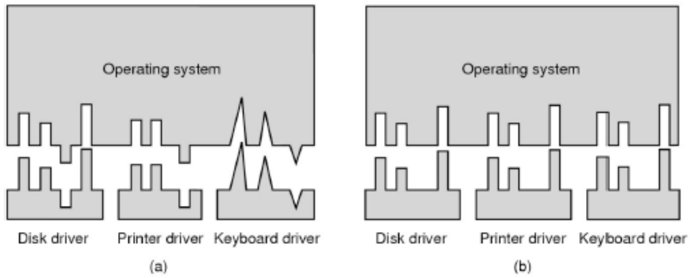
10.8.3. بقية نظام التشغيل المستقلة عن الأجهزة

البرامج المستقلة عن الأجهزة في نظام التشغيل تشمل:

- الواجهة مع التعريفات.
- الخازن المؤقت buffer.
- الإعلان عن الأخطاء.
- حجز وتحرير الأجهزة المخصصة.
- تقديم حجم كتل مستقل عن الأجهزة.

10.8.3.1. الواجهة الموحدة مع التعريفات

على نظام التشغيل توفير واجهة تمكن التعريفات من الإندماج مع نظام التشغيل وتمكن نظام التشغيل من استخدام التعريف، وعلى مصممي التعريفات معرفة كيف يكتبوا تعريفاتهم بحيث تشبك مع نظام التشغيل عبر هذه الواجهة.



شكل رقم (10-6): (b) الواجهة الموحدة لربط التعريفات مع نظام التشغيل.

(a) واجهات مختلفة لربط التعريفات مع نظام التشغيل.

من الأفضل أن تكون الواجهة موحدة لكل الأجهزة، فطريقة ربط تعريف الطابعة تكون شبيهة بطريقة تعريف سواقة القرص، تشبه طريقة تعريف المودم، تشبه طريقة تعريف الأجهزة الأخرى، أنظر الشكل (10-6).

10.8.3.2. الخازن المؤقت

الأجهزة بنوعيها الحرفية والكتلية تحتاج خازن مؤقت، حيث توضع فيه البيانات الواردة من الجهاز (في حالة جهاز دخل)ن أو الداخلة للجهاز (إذا كان جهاز خرج) ومن ثم إرسالها للجهة التي تطلبها. وجود الخازن يحسن الأداء بحيث يجعل التعامل مع الاجهزة أسرع.

قد يكون هنالك خازن واحد في مساحة المستخدم أو خازن واحد في مساحة النواة أو خازنين واحد في مساحة المستخدم وآخر فيمساحة النواة. والأخير هو الأفضل.

10.8.3.3. الإعلان عن الأخطاء

إذا حدث خطأ في جهاز، كأن نحاول الكتابة على جهاز دخل أو القراءة من جهاز خرج، فعلى نظام التشغيل إرسال تقرير بالخطأ للعملية أو للمستخدم والسماح

للمستخدم بإختيار ماذا يريد، هل يريد إعادة المحاولة أو تجاهل الخطأ أو إلغاء العملية (قتلها).

بعض الأخطاء الجسيمة تجعل نظام التشغيل إرسال تقرير بالخطأ ثم التوقف إجباريا.

10.8.3.4. حجز وتحرير الأجهزة المخصصة (غير قابلة للمشاركة)

هنا تقوم عملية واحدة فقط باستخدام الجهاز لذلك على نظام التشغيل حجزه في حالة ان مستخدم وتحريره عندما تفرغ العملية منه لتتمكن عمليات أخرى من استخدامه.

10.8.3.5 أحجام كتل موحدة

قد تختلف احجام الكتل بين الأجهزة، مثلا قد تكون هنالك أحجام قطاعات مختلفة بين الأقراص المختلفة. هنا على نظام التشغيل إخفاء الإختلاف وتقديم حجم كتلة موحد.

10.8.4. برمجيات الدخل والخرج في مستوى المستخدم

معظم برمجيات الدخل والخرج موجودة ضمن نظام التشغيل، وتوجد مكتبات تسمح للمستخدم التعامل مع هذه البرمجيات. فيستطيع المستخدم استدعاء نداء النظام المناسب من المكتبة للتعامل مع الجهاز الذي يريد.

10.9. القرص الصلب

سندرس القرص الصلب دراسة مستفيضة كدراسة حالة لأجهزة الدخل والخرج، ذلك لانه من أهم أجهزة الدخل والخرج ولا يخلو جهاز حاسب منه، وهو بالإضافة على ذلك يعتبر جهاز دخل وخرج في نفس الوقت.

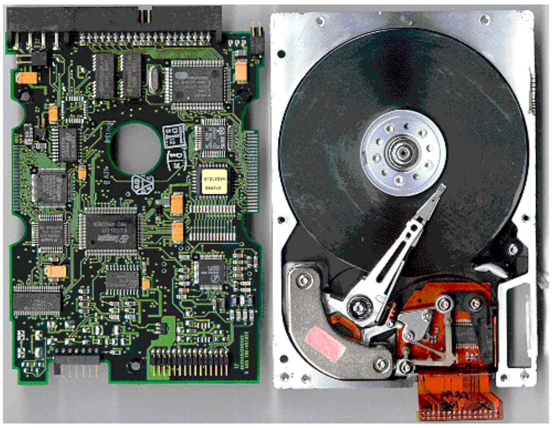
يتكون القرص الصلب مثله مثل أي جهاز من شقين :

- شق المتحكم وهو كرت يوجد باللوحة الأم غالبا ويتصل بشريط (ناقل بيانات) مع القرص الصلب.

- شق الجهاز وهو القرص الصلب، الشكل (7-10)، وهو يتكون من جزء ميكانيكي وجزء إلكتروني.



•



شكل (7-10): القرص الصلب.

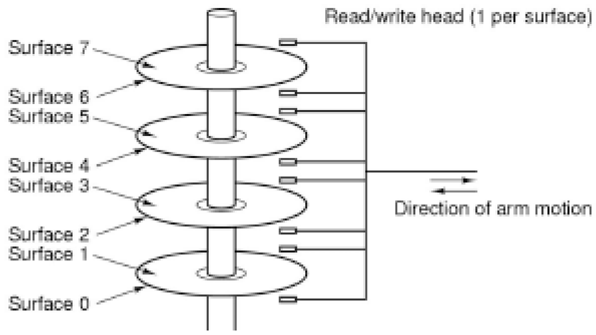
يخزن القرص الصلب البيانات في شكل مغناطيسي.

10.9.1. مكونات الجزء الميكانيكي

يتكون من أسطوانات مركبة على محور مشترك، وكل اسطوانة مكونة من سطحين، لكل سطح رأس قراءة وكتابة، ترتبط رؤوس القراءة والكتابة جميعها بذراع واحد بحيث عندما يتحرك الذراع يحرك معه كل الرؤوس، وحركة الذراع تكون أفقية مما يجعل رؤوس القراءة والكتابة تمر عبر مسارات الأسطح مع بعضها وتصل لنفس المسارات في كل الأسطح في نفس الوقت، شكل (8-10).

10.9.2. طريقة عمل القرص الصلب

لقراءة أو كتابة معلومة يتحرك الذارع الذي ترتبط به رؤوس القراءة والكتابة ليصل المسار المطلوب، هذه العملية تستغرق من 5 إلى 10 ملي ثانية.



شكل رقم (10-8): أجزاء القرص الصلب الداخلية.

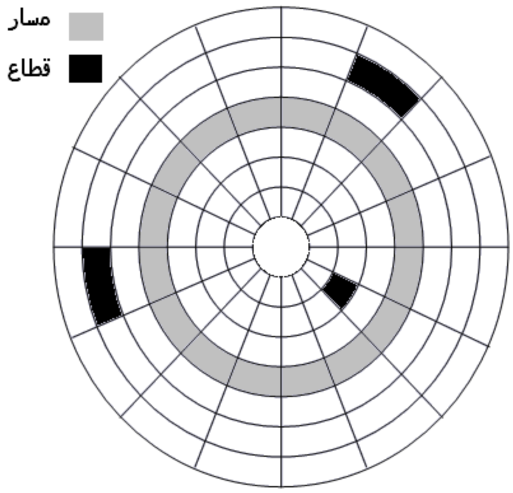
ثم تنتظر رؤوس القراءة والكتابة دوران الأسطوانات حتى تكون القواطع المطلوبة تحت رؤوس القراءة والكتابة، وتستغرق عملية الدوران هذه أيضا مدة تتراوح بين 5 و 10 ميلي ثانية.

ثم تبدأ رؤوس القراءة والكتابة بنقل البيانات في شكل كتل إلى ذاكرة المتحكم المرتبط بالقرص، حيث يتم ذلك بسرعة تتراوح بين 5 إلى 320 ميغابايت في الثانية.

هذه العوامل الثلاث تؤثر في سرعة التعامل مع القرص الصلب ككل.

10.9.3. الأجزاء الإلكترونية

عبارة عن لوح إلكتروني مهمته تحويل الإشارات الكهربائية (البيانات) إلى مناطق ممغنطة على القرص، ثم استعادتها متى ما طلب منه ذلك. كذلك يتحكم الجزء الالكتروني بالتحكم في دوران المحور (الأقراص) وحركة الذراع المثبت فيه رؤوس القراءة والكتابة.



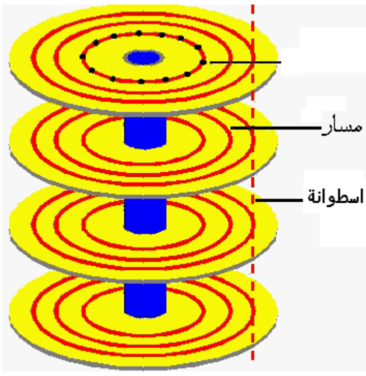
شكل رقم (10-9): تقسم الاسطوانة إلى مسارات وكل مسار إلى قطاعات.

10.9.4. المسار (tracks) والقطاع (sector)

يتم تقسيم كل أسطوانة إلى مسارات دائرية وكل مسار يقسم إلى أجزاء صغيرة متساوية في الحجم تسمى القطاعات، طول القطاع الواحد حوالي 512 بايت وهو أصغر وحدة قياس للتعامل مع القرص الصلب، الشكل رقم (10-9).

10.9.5. الاسطوانة (السلندر)

إن رؤوس القراءة والكتابة مثبتة في محور مشترك ومحرك واحد وبالتالي ستكون حركتها موحدة، فإذا كان واحد من الرؤوس على المسار الخارجي الأخير من قرص ما فإن الرؤوس الأخرى جميعها ستكون على المسار المشابه في الاقراص الأخرى. لذلك المسارات المتشابهة على كل الأسطح تكون أسطوانة. فمثلاً في الشكل (10-10) تكون المسارات الثمانية الخارجية أسطوانة. لذلك من الأفضل تخزين البيانات في شكل اسطوانات ذلك لأن رؤوس القراءة والكتابة تكون معا في مكان واحد وهذا يوفر وقت فلا نحتاج أن نحرك رؤوس القراءة والكتابة كذا مرة.



شكل رقم (10-10): المسارات المتشابهة في كل الأسطح تمثل أسطوانات.

مثال (10-1)

إذا كان لدينا خمسة أقراص والمسارات في كل سطح مرقمة من صفر إلى 202 فإن :

- عدد المسارات = 203
- عدد الأسطح = 10
- عدد رؤوس الكتابة والقراءة = 10
- عدد الأسطوانات = 203

مثال (10-2)

في المثال (10-1) إذا كان سعة المسار الواحد هي 40000 حرف أوجد:

- سعة الوجه (السطح) الواحد .
- سعة الاسطوانة الواحدة.
- السعة الكلية للقرص.

- عدد الممرات اللازمة لتخزين محتويات ملف مكون من 800000 حرف.
- عدد الاسطوانات اللازمة لتخزين محتويات هذا الملف.

الحل

- سعة الوجه الواحد = عدد المسارات × سعة المسار

203 x 40000 = 8120000

- سعة الاسطوانة = عدد مسارات الاسطوانة × سعة المسار = 400000
- السعة الكلية = سعة الاسطوانة × عدد الاسطوانات

203 x 400000 = 81200000

- عدد المسارات :

800000 / 40000 = 20

- عدد الاسطوانات : 10= 20/10

10.10. جدولة القرص

هنا يريد نظام التشغيل الوصول السريع للبيانات في القرص الصلب لتحقيق الكفاءة المطلوبة. وعملية الوصول للبيانات في القرص الصلب تعتمد على عوامل كثيرة مثل الزمن المستغرق لقراءة أو كتابة كتلة في القرص الصلب، وهذا أيضا يعتمد على ثلاث عوامل هي:

- زمن البحث (seek time).
- زمن دوران الاسطوانة لتصل الكتلة تحت رؤوس القراءة والكتابة (rotational latency).
- زمن نقل البيانات (transfer rate).

عندما يريد برنامج التعامل مع القرص الصلب، سيرسل طلبه لنظام التشغيل، حيث يحتوي الطلب على نوع العمل (قراءة ام كتابة) وعنوان المعلومة بالقرص وإلى أين نريد نقلها أو تخزينها. إذا كان المتحكم والقرص الصلب متاحين فسيتم تنفيذ الطلب على الفور، أما إذا كانا أحدهما أو كلاهما مشغولين، فسيتم وضع الطلب في صف انتظار الطلبات الموجه للقرص (requests pending queue). وعندما ينتهي القرص من انجاز هذا الطلب، يستطيع نظام التشغيل اختيار طلب آخر من صف الانتظار (إذا كان هنالك طلبات للتعامل مع القرص في الإنتظار).

طريقة اختيار الطلب التالي من صف الطلبات يعتمد على الخوارزمية التي يستخدمها نظام التشغيل. من الخوارزميات المستخدمة ما يلي:

- الأقدم أولا (First come first served).
- خوارزمية الزمن الأقل أولا (shortest seek time first).
- خوارزمية المصعد (elevator) أو المسح Scan.
- خوارزمية LOOK
- خوارزمية المسح الدائري (C-Scan).
- خوارزمية C-LOOK

وأفضل طريقة لتوضيح طريقة عمل هذه الخوارزميات هو عبر المثال التالي.

مثال (3-10)

إذا كان لدى صف من الطلبات (مرتبة حسب زمن وصولها)ن تريد التعامل مع بيانات بالاسطوانات التالية بالقرص الصلب:

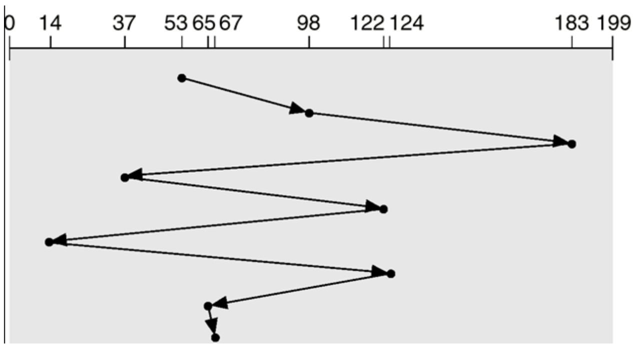
98, 183, 122, 37, 14, 124, 65, 67

وإذا كان رأس القراءة والكتابة الآن في الأسطوانة رقم 53. وضح كيف سيقوم نظام التشغيل بخدمة هذه الطلبات بجميع أنواع الخوارزميات أعلاه.

الحل

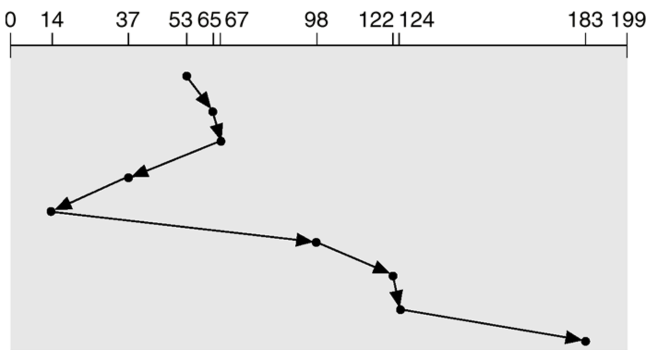
في خوارزمية FCFS سيتحرك رأس القراءة والكتابة لتلبية الطلبات (الأقدم أولا) حسب الترتيب التالي (نفس الترتيب الموجود في المسألة):

53, 98, 183, 122, 37, 14, 124, 65, 67



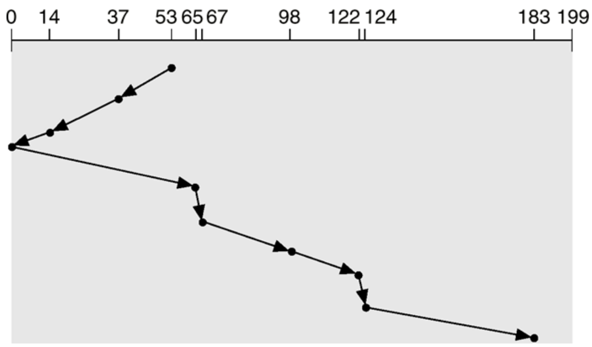
في خوارزمية SSTF سيتحرك رأس القراءة والكتابة لتلبية الطلبات حسب قربها من رأس القراءة والكتابة (الأقرب أولا)، حيث نقيس المسافة بين مكان رأس القراءة والكتابة وبين الطلبات، فنخدم الطلب الذي يعطي مسافة أقل. فيكون ترتيب خدمة الطلبات كما يلي:

53, 65, 67, 37, 14, 98, 122, 124, 183



في خوارزمية SCAN سيتحرك رأس القراءة والكتابة لتلبية الطلبات بدأ من مكان رأس القراءة والكتابة (متجها نحو البداية)، فيخدم كل اسطوانة يمر بها، ثم يعكس وجهته عندما يصل الصفر ليخدم كل اسطوانة يمر بها إلى أن يصل النهاية (يعمل بطريقة المصعد، لذلك أحيانا يطلق عليه خوارزمية المصعد)، وبالتالي فإن ترتيب خدمة الطلبات النهائي سيكون كالتالي:

53, 37, 14, 0, 65, 67, 98, 122, 124, 183, 199

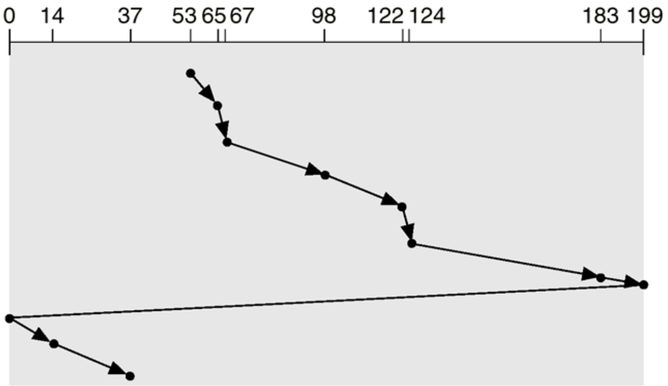


في خوارزمية LOOK سيتحرك رأس القراءة والكتابة لتلبية الطلبات بدأ من مكان رأس القراءة والكتابة (متجها نحو البداية)، فيخدم كل اسطوانة يمر بها إلى أن يصل أخر طلب قبل موجود في طريقه إلى البداية (ليس باالضرورة أن يصل الصفر)، ثم يعكس وجهته ليخدم كل الطلبات التي تقابله في طريق الرجعة إلى آخر طلب طلب موجود (ليس بالضرورة أن يصل النهاية). خدمة الطلبات ستكون كالتالي:

53, 37, 14, 65, 67, 98, 122, 124, 183

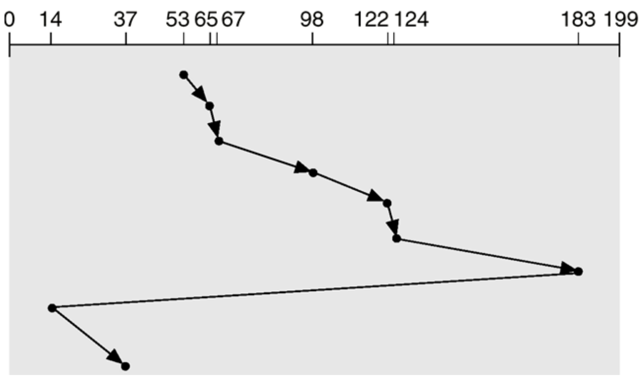
في خوارزمية C-SCAN سيتحرك رأس القراءة والكتابة لتلبية الطلبات بدأ من مكان رأس القراءة والكتابة (متجها نحو النهاية)، فيخدم كل اسطوانة يمر بها، عندما يصل إلى نهاية القرص يعكس إتجاهه راجعا ليبدأ من البداية (دون أن يخدم أي طلب في رجعته للبداية)، ليبدأ من الصفر ويخدم كل ما يمر به إلى أن يصل نهاية القرص. يشبه المصعد الذي يحمل الصاعدين فقط ثم يرجع فارغ ويحمل الصاعدين مرة أخرى وهكذا (مصعد اتجاه واحد). وهكذا. ترتيب خدمة الطلبات النهائي سيكون كالتالي:

53, 65, 67, 98, 122, 124, 183, 14, 37



في خوارزمية C-LOOK هي مطورة من C-SCAN، فهنا يتحرك رأس القراءة والكتابة لتلبية الطلبات بدأ من مكان رأس القراءة والكتابة (متجها نحو النهاية)، فيخدم كل اسطوانة يمر بها إلى أن يصل آخر طلب (قد يكون قبل نهاية القرص) ثم يرجع ليبدأ من البداية (دون أن يلبي أي طلب في رجعته للبداية)، ليبدأ من أول طلب موجود (ليس بالضرورة أن يصل إلى بداية القرص)، ويستمر مرة ثانية إلى أخر طلب بالقرص. ترتيب خدمة الطلبات النهائي سيكون كالتالي:

53, 65, 67, 98, 122, 124, 183, 14, 37

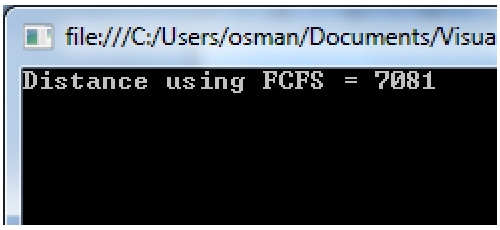


10.11.محكاة جدولة القرص

يمكن كتابة برنامج لحساب حركة الذراع بنفس الطريقة أعلاه. مثلا يمكننا وضع الطلبات في مصفوفة ثم نستخدم التكرار لحساب الفرق بين مكان الذراع الحالى والطلبات. مثلا لحساب مسافة الطلبات بخوارزمية FCFS، يمكن كتابة شفرة كالتالي:

```
Dim req(10), Current_Track, distance As Integer
req(0) = 86
req(1) = 1470
req(2) = 913
req(3) = 1774
req(4) = 948
req(5) = 1509
req(6) = 1022
req(7) = 1750
req(8) = 130
distance = 0
Current_Track = 143
For i = 0 To 8
distance = distance + Math.Abs(Current_Track - req(i))
Current_Track = req(i)
Next
Console.WriteLine("Distance using FCFS = " & distance)
Console.Read()
```

في البرنامج أعلاه وضعنا الطلبات في المصفوفة req ومكان رأس القراءة والكتابة الحالي في المتغير Current_Track، واستخدمنا تكرار for لحساب الفروقات بين الطلبات ووضعها في المتغير distance. فيما يلي شكل مخرج البرنامج.



10.12. تمارين محلولة

1. إذا كان لدينا قرص به 5000 أسطوانة مرقمة من 0 إلى 4999. حاليا يتم خدمة الأسطوانة رقم 143، والطلب السابق الذي تم خدمته كان في الأسطوانة 125. صف الطلبات التي نريد خدمتها حسب ترتيب وصولها هو:

86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

بداية من موقع رأس القراءة والكتابة الحالي (143)، ماهي المسافة التي يقطعها ذراع القرص في حركته لخدمة هذه الطلبات، باستخدام كل من الخوارزميات التالية:

- FCFS
- SSTF
- SCAN
- LOOK
- C-SCAN
- C-LOOK

الحل

(أ) باستخدام FCFS: ستتم خدمة الطلبات بالترتيب التالي:

143, 86, 1470, 913, 1774, 948, 1509, 1022, 1750, 130

مجموع المسافة المقطوعة هو 7081

(ب) باستخدام SSTF: ستخدم الطلبات بالترتيب التالي:

143, 130, 86, 913, 948, 1022, 1470, 1509, 1750, 1774

حيث سيكون مجموع المسافة المقطوعة هو 1745

(ج) باستخدام SCAN ستخدم الطلبات كما يلي:

143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 130, 86

بمجموع مسافة هو 9769

(د) خوارزمية LOOK تخدم الطلبات بالترتيب التالي:

143, 913, 948, 1022, 1470, 1509, 1750, 1774, 130, 86

بمجموع مسافة يساوي 3319

(هـ) خورازمية C-SCAN: تتم خدمة الطلبات كما يلي:

143, 913, 948, 1022, 1470, 1509, 1750, 1774, 4999, 0, 86, 130

حيث يكون مجموع المسافة المقطوعة تساوي 9813

(و) خوارزمية C-LOOK: خدمة الطلبات في هذه الخوارزمية يكون كالتالي:

143, 913, 948, 1022, 1470, 1509, 1750, 1774, 86, 130

بمجموع مسافة يساوي 3363

2. كيف تزيد DMA من كفاءة المعالج ؟ ذلك لأنها تجعل المعالج حرا من الإرتباط بمتابعة عمليات الدخل والخرج ويستطيع القيام بأعمال أخرى.

3. إذا كان لدينا المعطيات التالية:

* صف الطلبات : 23, 89, 132, 42, 187

* اسطوانات القرص: 200 أسطوانة مرقمة من 0 إلى 199

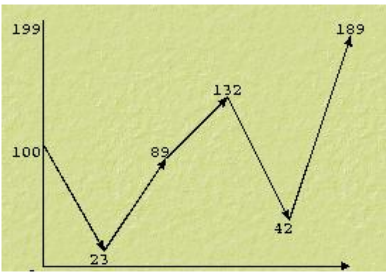
* رأس القراءة والكتابة: حاليا في 100

أحسب المسافة التي يقطعها ذراع القرص في حركته لخدمة هذه الطلبات، باستخدام كل من الخوارزميات التالية:

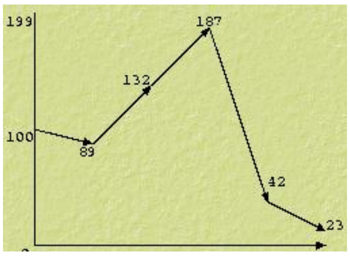
- FCFS
- SSTF
- SCAN
- LOOK
- C-SCAN
- C-LOOK

الحل

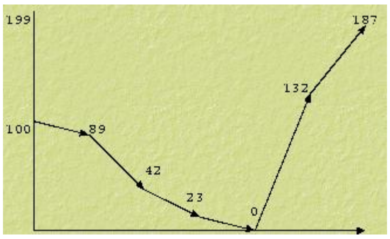
(أ) باستخدام FCFS: المسافة المقطوعة : 276



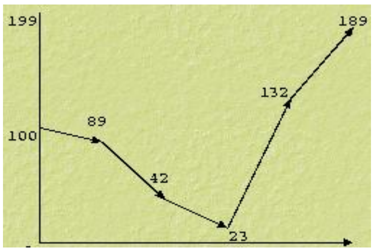
(ب) باستخدام SSTF: المسافة المقطوعة : 273



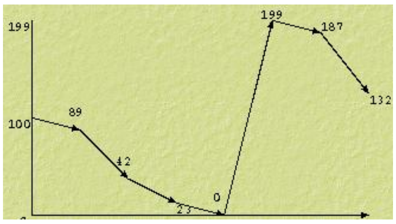
(ج) باستخدام SCAN : مجموع مسافة : 101



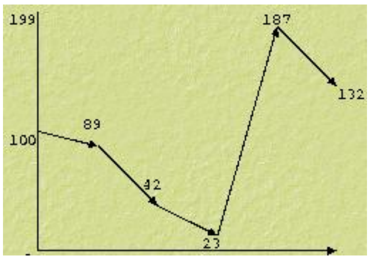
(د) خوارزمية LOOK مجموع مسافة : 241



(هـ) خورازمية C-SCAN: مجموع المسافة : 366



(و) خوارزمية C-LOOK: مجموع المسافة: 296



10.13. تمارين غير محلولة

1. يتكون الجهاز من شقين، ما هما ؟
2. هنالك نوع من الأجهزة أحدهما يتعامل بطريقة الكتل والثاني يتعامل بطريقة ___________؟
3. وضح طريقة عمل DMA ؟
4. أذكر خمس من خوارزميات جدولة القرص ؟
5. أذكر أهداف مدير الأجهزة ؟

الباب الحادي عشر:

مدير الملفات

الباب الحادي عشر

مدير الملفات (*Files Manager*)

يعتني مدير الذاكرة بالبيانات والمعلومات أثناء وجودها بالذاكرة الرئيسية ويسمى هذا النوع من التخزين، التخزين قصير المدى (short-term storage)، ولكننا غالبا سنحتاج لحفظ معظم المعلومات لفترة طويلة أو ما يسمى بالتخزين طويل المدى (long-term storage). التخزين طويل المدى يكون في ذاكرة ثانوية تحتفظ بمحتوياتها لأيام وليالي، هذه الأجهزة (مثل القرص الصلب والقرص المرن والأقراص الضوئية والفلاش) يتعامل معها نظام التشغيل بطريقة موحدة هي الملف (file). يقوم مدير الملفات كجزء من نظام التشغيل بعمليات تخزين واسترجاع الملفات في هذه الأجهزة (أجهزة التخزين الدائم).

بدون مدير الملفات ستكون بياناتنا بأنواعها سواء كانت نصوصا أو برامجا أو أصواتا، مخزنة في شكل أصفار ووحائد (أرقام ثنائية)، ولن نستطيع التمييز بينها فهي مخزنة في مكان واحد دون حدود واضحة بينها، فلن نعرف أين بداية الملف ولا نهايته، ولن نعرف نوعه ولا طرق حمايته.

يقوم مدير الملفات بكل تفاصيل التخزين الدقيقة نيابة عنا، فهو يعرف نوع الملفات ومكانها بالقرص وكيف يخزنها وكيف يسترجعها، وكيف يحذفها وما إلى ذلك (التعامل الحقيقي مع الملفات). بينما يوفر للمستخدم واجهة منطقية تمكنه من التعامل مع الملفات بصورة ميسرة، فهو يعرف نوع الملف من خلال ايقونة ويحفظ الملف باسم واضح ومفهوم، ويحذف ويعدل وينشئ الملفات دون أن يعرف أين تم تخزينها وفي أي مقاطع أو مسارات وضعت.

11.1. أهداف إدارة الملفات

هي القيام بكل ما يتعلق بالتعامل مع الملفات، مثل:

- إنشاء وحذف الملفات.
- إخفاء تفاصيل مكان الملف في القرص.

- عمليات الوصول إلى الملفات من قراءة وكتابة.
- توفير مشاركة الملفات.
- توفير الحماية على الملفات.
- توفير الاعتمادية بعمليات النسخ الاحتياطي.

11.2. تعريف الملف

الملف هو مجموعة من المعلومات ذات علاقة وبنية منطقية، فالوثيقة مثلا تتألف من كلمات وسطور وفقرات وصفحات (بنية منطقية) وتحتوي على موضوع واحد (معلومات ذات علاقة).

الملفات قد تكون حرة البنية (غير مهيكلة) مثل ملفات النصوص ، حيث يتكون الملف من بايتات وحروف وسطور، وقد تكون مهيكلة مثل ملفات قواعد البيانات التي تتكون من حقول وسجلات.

يدعم نظام التشغيل ينكس الملفات غير مهيكلة والتي تكون عبارة عن سلسلة من البايتات والحروف.

11.3. صفات الملف (*File Attributes*)

اسم الملف عادة هو سلسلة من الحروف مثل (example .doc) بعض نظم التشغيل مثل ينكس تميز بين الحروف الكبيرة والصغيرة في الاسم مثلاً (Example .doc) غير الاسم (example .doc).

و لكل ملف صفات مثل :

- الاسم (name).
- النوع (type).
- الموقع (location).
- الحجم (size).

- الحماية (protection) .
- كلمة المرور.
- المنشئ (creator).
- المالك (owner).
- الزمن والتاريخ والمستخدم (time , data , and user identification) مثلاً (زمن وتاريخ الإنشاء ، زمن وتاريخ آخر تعديل).

11.4. العمليات على الملفات

هنالك عمليات مختلفة يمكن أن تنفذ على الملفات تختلف باختلاف النظم، نذكر منها:

- إنشاء الملف (create a file).
- فتح ملف.
- الكتابة في ملف.
- القراءة من ملف .
- إضافة بيانات في نهاية ملف موجود (append).
- البحث عن معلومة في ملف (seek).
- حذف ملف.
- تفريغ ملف (مسح محتوياته).
- معرفة صفات ملف (get attributes).
- إضافة أو تعديل صفات ملف (set attributes).
- تغيير اسم ملف.

11.5. أنواع الملفات

نوع الملفات	الامتداد	المهمة
تنفيذي (executable)	Exe ,com , bin	جاهز للتنفيذ
object	Obj , o	تم ترجمته لكن يحتاج ربطه مع مكتبات أخرى حتى يصبح تنفيذي
مصدري (source)	.java, .c, .cs,…	برنامج مكتوب بلغة برمجة معينة لكن لم يتم ترجمته بعد
حزمة (Batch)	Bat , sh	مجموعة من أوامر نظام التشغيل تجمع في ملف (مثل (dir, cls
Text	Txt , doc	وثائق نصية كالتي تكتب على وورد والمفكرة.
أرشيف (Archive)	Are ,zip , tar	مجموعة من لملفات يتم ضغتها في ملف واحد

11.6. طرق الوصول ***access method***

- يمكن تسلسلي أو تتابعي (sequential access).
- وصول مباشر (direct access).

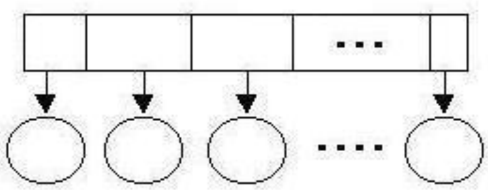
11.7. بنية الدليل ***Directory structures***

يتم تقسيم نظام الملفات إلى أقسام (partitions) أحياناً تسمي (minidisks) أو (volumes)، وكل قسم (partitions) يحتوى معلومات عن الملفات المخزنة به.

لتنظيم الملفات ووضع المتشابه منها في صورة منظم نستخدم ما يسمى بالدليل أو المجلد، حيث كل مجلد يحتوي على مجموعة من الملفات. العمليات التي تجرى على المجلدات كثيرة وتشبه تلك التي تجرى على الملفات، بل يعتبر المجلد بحد ذاته ملف.

11.7.1. مستوى الدليل الواحد Single – level directory

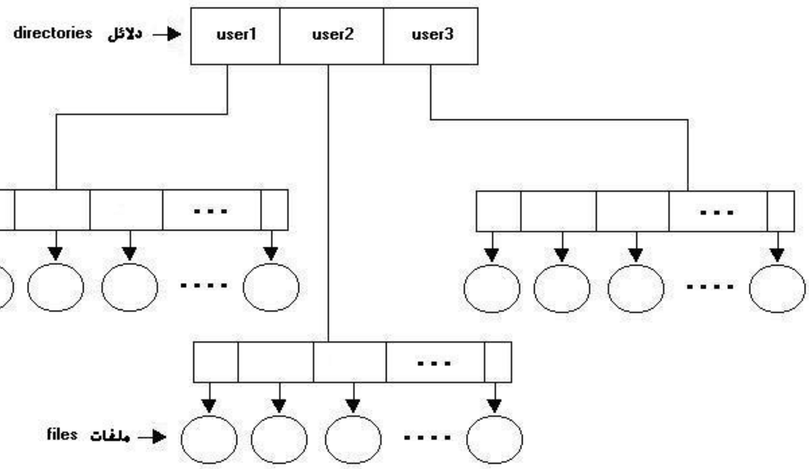
أبسط نوع وفيه تكون كل الملفات محفوظة في مكان (مجلد) واحد، لذلك لا يمكن تسمية ملفين باسم واحد، الشكل (7-).



شكل رقم (11-1): مستوى الدليل الواحد.

11.7.2. مستوى الدليلين Two – level directory

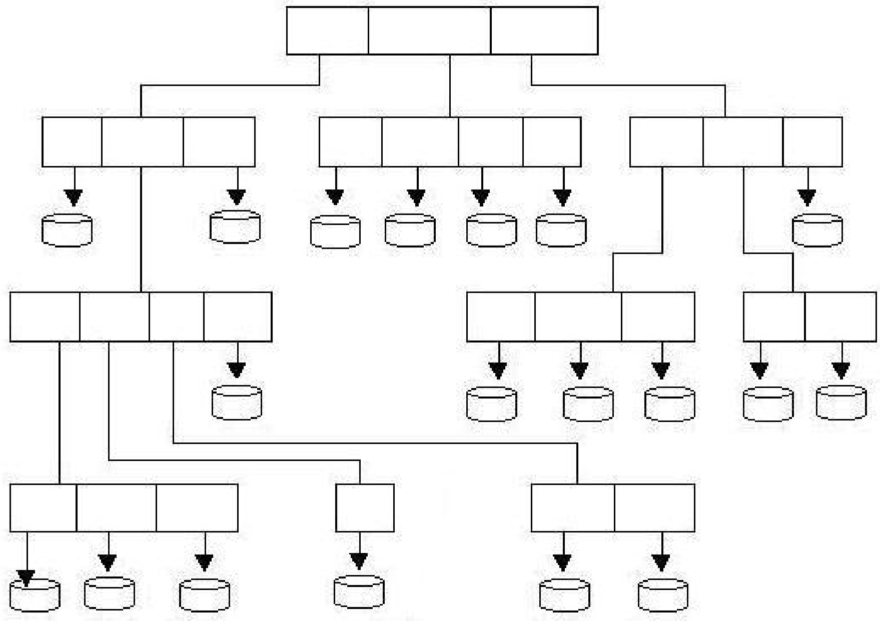
كل مستخدم لديه دليل خاص به، وبالتالي كل مستخدم يمكنه استخدام أسماء حتى ولو كانت مستخدمة عند الآخرين (يمكن تكرار اسم الملف لكن كل اسم في دليل مختلف.



شكل رقم (11-2): مستوى الدليلين.

11.7.3. الدليل الشجري Tree structured directory

في الدليل الثنائي يستطيع كل مستخدم تسمية ملفاته كما يريد حتى ولو كانت موجودة أسماء مثلها عند المستخدمين الآخرين، ولكن ماذا لو أراد تسمية ملفين باسم واحد أو تجميع كل ملفات ذات صلة في دليل لوحدها، أكيد لن يستطيع فعل ذلك في الدليل الثنائي، لذلك جاء الدليل الشجري لحل مثل هذه المشاكل، فكل مستخدم له مطبق الحرية في بناء ما يرد من مجلدات ولأي درجة من المستويات وبالتالي يستطيع تنظيم مجلدات وملفات بطريق مختلفة وبهرمية مختلفة كما يردي ووقت ما يشاء. وهذا هو النظام المتبع حاليا في معظم نظم التشغيل الحديثة.



شكل رقم (11-3): الدليل الشجري أو الهرمي.

11.8. الحماية

عند حفظ المعلومات في الحاسب عليك العمل على حمايتها خاصة إذا كانت هامة وسرية. ولكن ممن أحميها ؟ من:

- الأعطال (damage)، لتوفير الاعتمادية والاستمرارية (reliability).
- التطفل (الوصول غير مسموح به (improper access))، لضمان سريتها وخصوصيتها.

11.8.1. النسخ الاحتياطي

أحد حلول الحماية من الأعطال هو عمل نسخ احتياطي دوري للمعلومات الهامة. مثلا في النظام البنكي ستكون بيانات العملاء وأرصدتهم والعمليات التي قاموا بها من سحب و إيداع، هامة جدا ولابد من عمل نسخ احتياطي لها بصورة دورية. قد يتم النسخ بواسطة فني أو بواسطة برامج صمم ليقوم بذلك تلقائيا.

توفر بعض نظم التشغيل خدمات النسخ الاحتياطي لجزء من القرص أو القرص كاملا، مثلا ويندوز توفر برنامج يقوم بعملية النسخ والاسترجاع يسمى Backup يوجد في accessories داخل القائمة System tools، شكل رقم (4-11). حيث يستخدم هذا البرنامج لتخزين الملفات الموجودة في القرص إلى وحدة تخزين أخري. ويمكن استرجاع النسخة الاحتياطية إلى القرص مرة أخرى بنفس البرنامج (restores).



شكل رقم (4-11): برنامج النسخ الاحتياطي في ويندوز XP.

عملية النسخ الاحتياطي تضمن على الأقل سلامة البيانات إذا تعطل القرص الصلب الذي يحويها، فنستطيع مثلا استبدال القرص المعطوب بآخر جديد واسترجاع بياناتك من مكان النسخ الاحتياطي.

7.8.2. أذونات الوصول

الحماية من التطفل تتم بطرق شتى منها أذونات الوصول. حيث يوفر نظام التشغيل حماية ضد أنواع الوصول من قراءة، كتابة، تنفيذ،... الخ. تتم الحماية عادة بواسطة أذونات الوصول (access permissions) التي تمنع أو تسمح للمستخدمين من الوصول إلى الملفات المحمية.

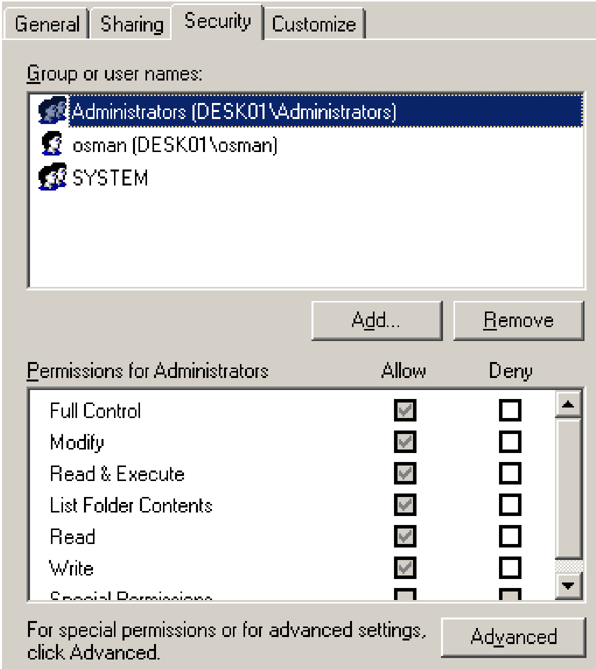
قوائم الوصول تحدد أنواع المستخدمين الذين يمكن منحهم أو منعهم الوصول إلى الملفات، مثل:

- المالك Owner.
- مجموعة معينة Group.
- الكل Universe.

إذ أن أذونات الوصول تحدد نوع الوصول إلى الملفات وقوائم الوصول تحدد من سيصل.

7.8.2.1. أذونات الوصول في ويندوز XP

إذا أردت عمل مشاركة لمجلد في ويندوز XP ستظهر نافذة بها تبويب يسمى Security يحتوي على قوائم وصول (group or user name) وأذونات وصول لكل نوع من أنواع قوائم الوصول(permissions)، مثلا إذا نقرت على Administrator في قوائم الوصول سيظهر في نافذة أذونات الوصول، الأذونات التي يمكن منحها للمشرف (permissions for Administrators) مثل التحكم الكامل (full control) أو القراءة والتنفيذ (Reaa & Execute)، وما إلى ذلك من أذونات تظهر في الشكل (11-5).



شكل رقم (5-11): أذونات الوصول في ويندوز XP.

7.8.2.2. أذونات الوصول في أوبونتو

كل شيء في ينكس ولينكس يعتبر ملف. وكل ملف لديه أذونات تسمح أو تمنع الوصول إليه. هنالك ثلاث أنواع من الوصول للملف هي القراءة (r أو 4)، الكتابة (w أو 2)، والتنفيذ (E أو 1). وهنالك ثلاث أنواع من المستخدمين هم:

المستخدم	*ls* output
المالك (owner)	`-rwx------`
مجموعة ما (group)	`----rwx---`
أخرى (other)	`-------rwx`

مثلا يمكنني معرفة الحماية الموجودة على الملف hosts بالأمر التالي:

ls -l /etc/hosts

فيكون ناتج تنفيذ الأمر كما يلي:

-rw-r--r-- 1 root root 288 2005-11-13 19:24 /etc/hosts

تفسير الناتج أعلاه:

-rw-r--r-- تفسر كما يلي:

owner = Read & Write (rw-)

group = Read (r--)

other = Read (r--)

أي أن المالك لديه صلاحيات القراءة والكتابة، المجموعة لديها صلاحية القراءة فقط، الآخرين لديهم صلاحية القراءة فقط.

نستطيع تغيير نوع الحماية باستخدام الأمر chmod والتي تكون صيغته كالتالي:

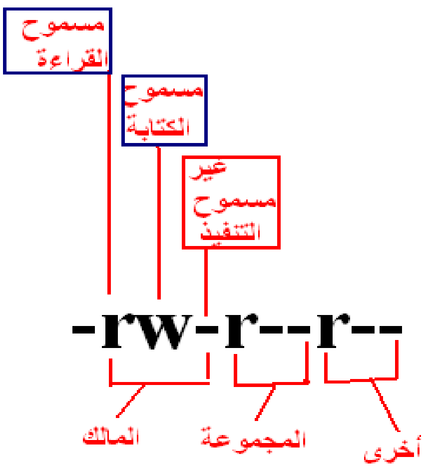
Usage: chmod {options} filename

حيث Options يمكن اختيارها من الجدول (1-7).

Options	Definition
u	owner
g	group
o	other
x	execute

write	w
read	r
add permission	+
remove permission	-
set permission	=

جدول (11-1): خيارات تغيير نوع الوصول.



شكل رقم (11-6): حروف الحماية.

11.9. طرق التخزين

مرونة الوصول المباشر تسمح لنا بتخزين معلوماتنا في أي مكان بالقرص، ولكن المشكلة التي تظهر هنا هي كيف نحجز مكان في القرص لتخزين هذه المعلومات؟ هنالك طرق مختلفة لحجز وتخزين ملفاتنا في القرص، يسعى مدير الملفات هنا أن يخزن ويسترجع البيانات بكفاءة عالية تقلل الزمن المستغرق في ذلك.

11.9.1. جدول الحجز

يستخدم مدير الملفات جدول كقاعدة بيانات تخزن فيها معلومات الملفات التي توضح مكان الملف بالقرص، مثل اسم الملف ومكان الملف بالقرص. عندما يطلب مستخدم ما، من نظام التشغيل فتح ملف معين، سيبحث مدير الملفات عن اسم الملف في جدول الحجز ثم يستخدم المعلومات الموجودة بالجدول لاسترجاع الملف.

إذا فقد مدير الملفات جدول الحجز لن يستطيع معرفة أماكن الملفات ولا أسماءها ويصبح استرجاعها مشكلة كبيرة.

مثال لجدول الحجز، جدول حجز الملفات ((File Allocation Table (FAT) في نظام تشغيل DOS. و FAT32 في نظام التشغيل ويندوز.

11.9.2.وحدة تخزين الملف

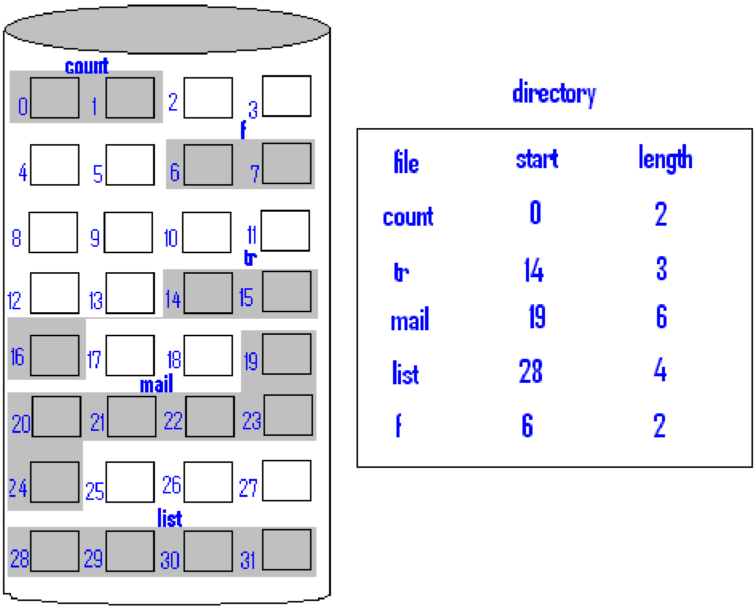
أصغر وحدة في القرص الصلب هي القاطع وطوله حوالي 512 بايت، لكنها لا تستخدم لتخزين الملفات لصغر حجمها، فإذا استخدمت كوحدة لتخزين الملفات سيؤثر هذا على الأداء. لذلك عادة تستخدم وحدة أكبر للتخزين تسمى تجمع (cluster)، حيث يتراوح حجم التجمع الواحد بين 2048 بايت إلى 32768 بايت أي ما يعادل 4 إلى 64 قاطع.

هنالك طرق مختلفة لتخزين الملفات سنتطرق لبعضٍ منها. سنستخدم كلمة كتلة للدلالة على القطاع أو التجمع (cluster).

11.9.2.1. تخزين متتال (contiguous allocation)

هنا تخزن الملفات في كتل متتالية. فإذا أراد نظام الملفات تخزين ملف يحتاج 5 كتل، فعليه البحث عن 5 كتل فارغة (متتالية) لتخزين الملف. زمن الوصول للقرص في هذه الطريقة سريع لأن رأس القراءة والكتابة بالقرص لن يحتاج إلى حركة. ولكن المشكلة تكمن في وجود فراغات متباعدة تنتج بسبب الحذف والتخزين الكثير للملفات،و لا يمكن الإستفادة من هذه الفراغات لأنها غير متتالية.

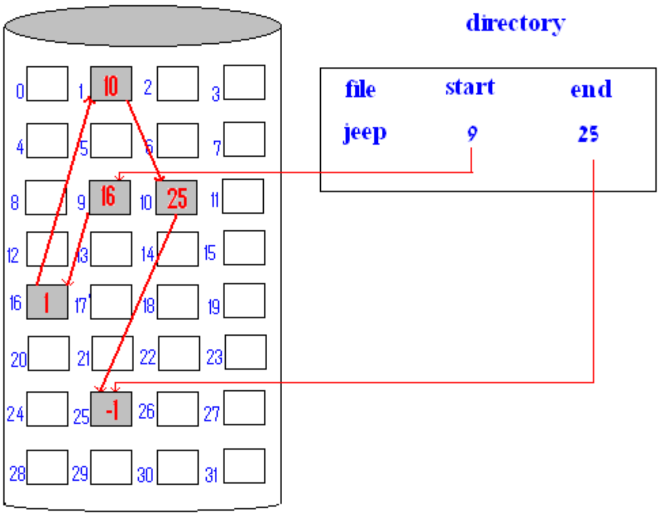
عندما يقوم مدير الملفات بتخزين الملف في القرص (في كتل متتالية)، سيخزن معلومات الملف في جدول الحجز بوضع عنوان أول كتلة تم تخزين الملف بها، وطول الملف (عدد الكتل التي يحجزها) مع اسم الملف، الشكل (11-7).



شكل رقم (11-7): تخزين متتالي.

11.9.2.2. تخزين رابطي (linked allocation)

هذه الطريقة تعالج مشاكل طريقة التخزين المتتالي. حيث يخزن كل ملف في كتل مرتبطة مع بعضها، حيث تشير كل كتلة إلى الكتلة التي تليها، بينما تؤشر آخر كتلة في الملف إلى -1. قد تكون الكتل موزعة في أماكن متباعدة داخل القرص (لا توجد فراغات خارجية). عند تخزين الملف في القرص يسجل مدير الملفات معلومات الملف في جدول الدليل (اسم الملف، وعنوان أول كتلة)، الشكل (11-8).



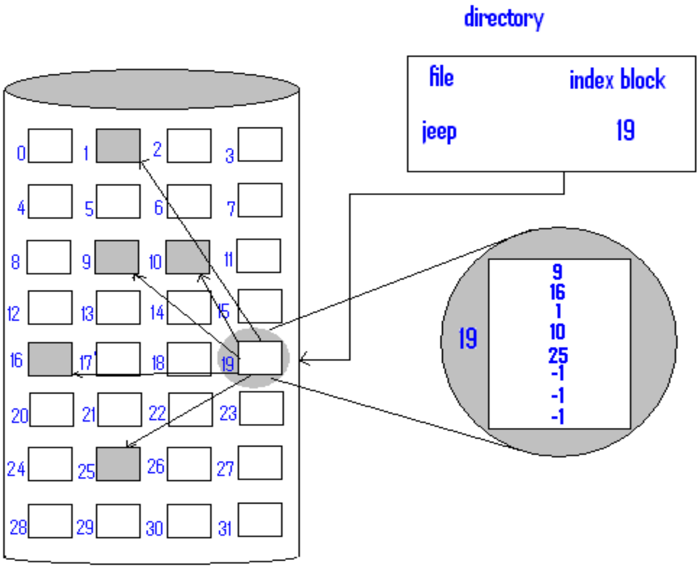
شكل رقم (11-7): تخزين رابطي.

هذا النوع يستخدم في نظام التشغيل DOS ويسمى File Allocation Table (FAT).

11.9.2.3. تخزين فهرسي (indexed allocation)

حل التخزين الرابطي مشكلة الفراغات الغير مستفاد منها في التخزين المتتالي، ولكن المشكلة فيه أن المؤشرات منتشرة في الكتل، حيث تحتوي كل كتلة على مؤشر للكتلة التي تليها، وبالتالي لن نستطيع إلى كتلة معينة مباشرة وإنما علينا المرور على كل الكتل واحدة تلو الآخرة لنصل لكتلة معينة مما يؤثر على الأداء.

حل التخزين الفهرسي هذه المشكلة وذلك بوضع عناوين كل الكتل في كتلة واحدة تسمى كتلة الفهرس. فيكون لكل ملف يخزن بالقرص كتلة تخزن فيها عناوين بقية الكتلة التي يستخدمها الملف، شكل (11-8).



شكل رقم (11-8): التخزين الفهرسي.

النظام الفهرسي لا يوجد به فراغات، ليس لديه مشكلة للوصول مباشرة إلى أي كتلة، ولكنه يحجز كتلة أو أكثر ليخزن فيها عناوين الكتلة الأخرى.

11.9.2.4. ما هي الطريقة المناسبة

اختيار طريقة الحجز التي ستطبق في نظام التشغيل تعتمد على الكفاءة وسرعة الوصول، ولكن هنالك عوامل أخرى تؤثر في اختيارنا على طريقة الوصول المناسبة مثل طريقة استخدام النظام.

تمارين

1. ما هي أهداف مدير الملفات ؟
2. عرف الملف ؟
3. أذكر خمس من صفات الملفات ؟
4. أذكر خمس من العمليات التي تتم على الملفات ؟
5. تتم حماية الملفات من أمرين، ما هما ؟
6. كيف نحمي ملفاتنا من الأعطال ؟
7. كيف نحمي ملفاتنا من التطفل ؟
8. أذكر ثلاث من طرق التخزين ؟
9. ما هو جدول الحجز allocation table، وما هي أهميته ؟
10. أبحث في الإنترنت عن أنواع جداول الحجز Fat و FAT32 و NTFS وقارن بينها ؟

الباب الثاني عشر:
النظم الموزعة

الباب الثاني عشر

النظام الموزع (*Distributed System*)

12.1. مقدمة

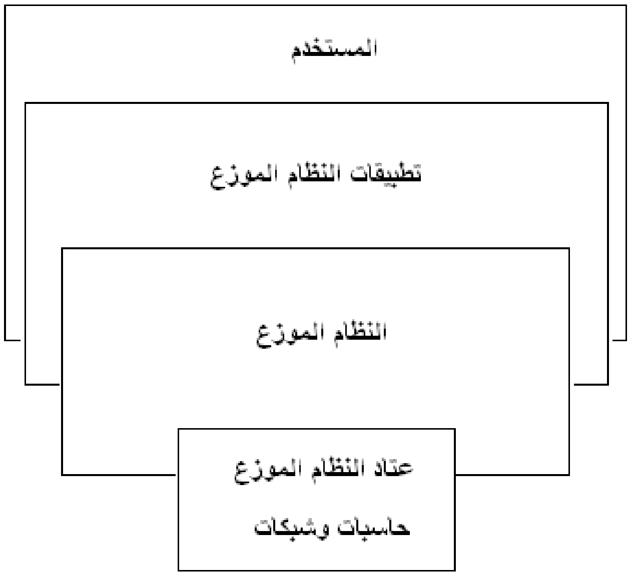
يتضح شكل أي نظام من خلال برمجياته بينما يظل العتاد خلف الكواليس كالجندي المجهول يعمل بلا كلل ولا ملل. نظم التشغيل لا تشذ عن هذه القاعدة فهي توفر الواجهة التي من خلالها نستخدم العتاد، فهو وسيط بيننا وبين العتاد .

قاعدة أخرى تقول إنه كلما تعقدت البرمجيات وبُذِل فيها الجهد الكبير كان الناتج برمجيات متميزة وسهلة الاستخدام. كذلك الجهد المبذول لبناء نظم موزعة يساهم في تيسير بناء التطبيقات الموزعة. فسهولة الاستخدام هدف أساسي وراء تصميم النظم الموزعة. أيضاً الشفافية التي نخفي بها تباين وتعقيدات العتاد (من أجهزة وشبكات) هي هدف يساعد في إزالة صعوبة وتعقيدات التعامل مع الأنظمة الموزعة.

> بناء تطبيق موزع من غير وجود نظام موزع، يشبه إلى حدٍ ما استخدام الحاسب من غير نظام تشغيل.

النظام الموزع هو طبقات برمجية توفر للمستخدم بيئة سهلة لبناء تطبيقات موزعة.

يمكن أن يكون النظام الموزع نظام تشغيل قائماً بذاته فهو هنا يقوم بدور نظام التشغيل العادي بالإضافة لدوره كنظام موزع يوفر الشفافية للتطبيقات الموزعة للتعامل مع العتاد كأنه جهاز واحد. وقد ينبني النظام الموزع كطبقة فوق نظام تشغيل وهو بهذا يكمل النقص الموجود بنظام التشغيل ويوفر الاثنان معاً بيئة سهلة لبناء تطبيقات موزعة.



شكل رقم (12-1): موقع النظام الموزع.

12.2. مهام برمجيات النظام الموزع

تقوم برمجيات النظام الموزع بالكثير من المهام مثل:

- إدارة عتاد النظام الموزع لتمكين المستخدمين وتطبيقاتهم من التشارك في الموارد (resources) ، مثل المعالجات ، الذاكرة ، الأجهزة الطرفية وعتاد الشبكة.
- توفير التشارك في البرامج والبيانات بأنواعها.
- إخفاء تفاصيل العتاد وتباينه بتوفير حاسب واحد افتراضي.

12.3. أنواع نظام التشغيل

ما رأيك لو أهديناك حاسباً بدون ويندوز مثلا، أو بدون لينكس ؟ بالتأكيد سترميه على قارعة الطريق إن لم تجد له نظام تشغيل. فنظام التشغيل أساس كل شي في الحاسب سواء كان هذا الحاسب شخصياً، أو مركزياً، أو نقال، أو مجموعة حاسبات مرتبطة بشبكة.

لا يستطيع المستخدم العادي التعامل مع الحاسب في غياب نظام التشغيل، وعلى المبرمج الكبير والمهندس القدير بذل الوقت الوفير لتنفيذ برنامج صغير (في غياب نظام التشغيل). إذن هدف نظم التشغيل الأساسي سهولة استخدام العتاد.

كذلك لا تستطيع تطبيقاتنا العمل على الحاسب في غياب نظام التشغيل فهو مهم لها كأهميته للمستخدم. فلا التطبيقات ولا المستخدم يستطيع التعامل مع العتاد مباشرة.

يمكن تقسيم نظم التشغيل إلى نوعين هما:

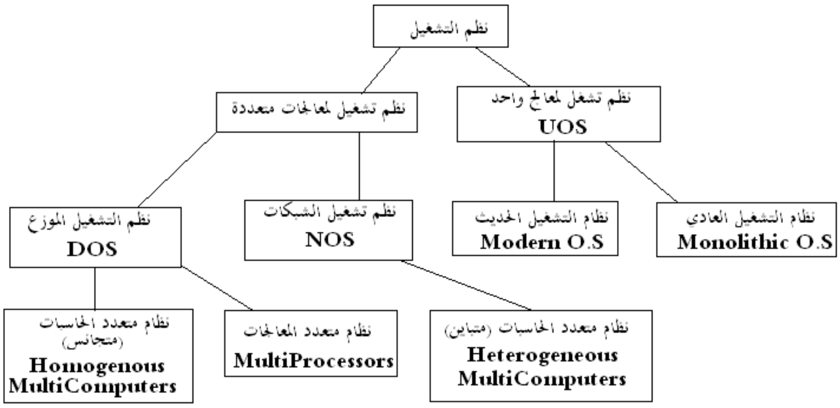
- نظام تشغيل معالج واحد.
- نظام تشغيل أكثر من معالج.

نظام تشغيل المعالج الواحد ينقسم بدوره إلى نوعين هما :

- تقليدي (monolithic).
- حديث (modern).

نظام التشغيل الذي يدير عدة معالجات ينقسم إلى:

- نظام تشغيل موزع (DOS).
- نظام تشغيل شبكات (NOS).



شكل رقم (12-2): أنواع نظم التشغيل

12.3.1. نظم تشغيل المعالج الواحد

يقوم نظام التشغيل هنا بإدارة عتاد حاسب به معالج واحد. وتوفير بيئة سهلة للمستخدم وتطبيقاته للإستفادة من العتاد بصورة مثلى.

الهدف الأساسي وراء نظام التشغيل هو إدارة عتاد الجهاز والسماح للمستخدمين من الاستفادة القصوى منه والتشارك فيه. تشارك التطبيقات في المورد يتم بتعدد المهام (multitasking). حيث يتم تحويل المورد بين التطبيقات بسرعة (كل تطبيق يأخذ نصيباً زمنياً في استخدام المورد)، وكلما زادت التطبيقات قل نصيب كل تطبيق في استخدام المورد.

من أهم متطلبات مشاركة الموارد هو حماية التطبيقات من بعضها البعض. فمثلا لو كان لدينا تطبيقان "أ" و "ب" يعملان معاً في نفس الوقت وفي نفس الذاكرة ، فلا يُسمح للتطبيق "أ" أن يتعدي على بيانات التطبيق "ب" والعكس. فتعمل التطبيقات ضمن الحدود والصلاحيات التي يوفرها لها نظام التشغيل ولا تتعداها. أيضاً لا يسمح للتطبيق بالوصول مباشرة للعتاد وإنما يتم ذلك عن طريق استدعاء دوال بنظام التشغيل تسمى نداءات النظام (system calls).

نظام التشغيل سكرتير العتاد فلا تصل التطبيقات للعتاد بدون أخذ مواعيد وموافقة منه.

إذا نظرنا لهيكل نظام التشغيل نجده مكوناً من طبقتين رئيسيتين هما :

- قلب نظام التشغيل أو نواته (Kernel).
- مساحة المستخدم (User) أو الغلاف أو الواجهة.

12.3.1.1. النواة

هي البرامج التي تقوم بإدارة العتاد من معالج وذاكرة وطرفيات وأقراص وغيرها. وهي التي تتحمل بالذاكرة الرئيسية عند فتحك للحاسب وهي التي تشغل وتدير باقي التطبيقات. وهي لطبيعة عملها تستطيع الوصول لأي جزء من العتاد وتنفيذ أي أوامر دون أن يمنعها أو يحدها شيء (هنا مسموح بتنفيذ كل أنواع الأوامر بما فيها التي تتعامل مباشرة مع جميع أجزاء الذاكرة وجميع المسجلات).

12.3.1.2. مساحة المستخدم (الغلاف)

هي الواجهة التي يتعامل معها المستخدم والتطبيقات، ومن خلالها نستطيع تنفيذ تطبيقاتنا، لذلك أحيانا تسمى واجهة المستخدم أو وضع المستخدم أو مساحة المستخدم. وهنا لا يسمح بالوصول المباشر للعتاد، فلا يمكن الوصول للذاكرة التي يقبع فيها نظام التشغيل مثلاً، ولا يمكن التعامل مع كل المسجلات. فيحدد نظام التشغيل الجزء المسموح الوصول إليه من الذاكرة والمسجلات التي يمكن للتطبيقات الوصول إليها (يمنع الوصول مباشرة إلى مسجلات الأجهزة (device registers)).

عند ما ينفذ المعالج برامج نظم التشغيل يكون في وضع النواة ، ويتغير لوضع المستخدم عند تنفيذ برامج المستخدم.

التحول من وضع النواة إلى وضع المستخدم يتم عبر نداءات النظام (system calls). حيث توفر هذه النداءات خدمات التعامل مع العتاد للتطبيقات. فمثلا إذا أراد تطبيق قراءة بيانات من ملف مخزن بالقرص، فإنه سينادي دالة بنظام التشغيل تقوم باحضار المعلومات من القرص.

جعل النواة تحتوي على كل برامج إدارة العتاد مثل برامج جدولة المهام ، برامج إدارة الذاكرة ، برامج نظام الملفات والتعامل مع الأقراص وغيرها من البرامج يسبب مشاكل. فالنواة ستكون:

- كبيرة.
- مستقرة في الذاكرة.
- في مكان واحد بالذاكرة.

في هذه الحال، يصبح تعديل جزء من نظام التشغيل صعباً ومعقداً ويتطلب الآتي:

- إجراء التعديل
- إغلاق الجهاز
- إعادة ترجمة نظام التشغيل (recompile).
- إعادة تنصيب نظام التشغيل (setup)

هذا النوع من نظام التشغيل نطلق عليه نظام التشغيل الرتيب (monolithic operating system). ويعتبر غير جيد لأنه يفتقر للآتي:

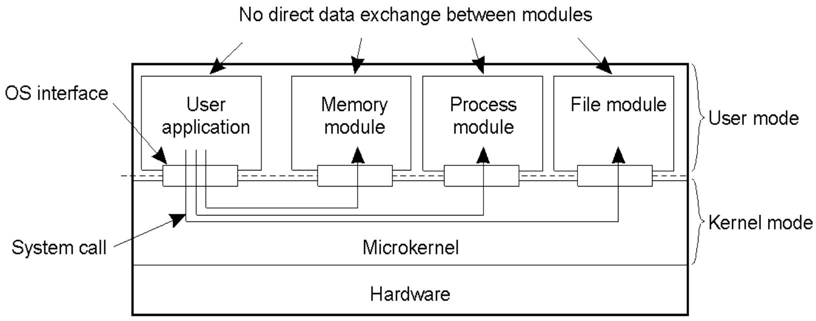
- الانفتاح (openness).
- الاعتمادية (reliability).
- الصيانة (maintainability).

12.3.2. نظام التشغيل الحديث (modern operating system)

يمكن تغيير نظام التشغيل ليصبح أكثر مرونة وذلك بتعديل النواة في نظام التشغيل العادي، فيصبح النظام الحديث كالآتي:

- تحويل البرامج التي تدير العتاد من النواة إلى مساحة المستخدم.
- جعل البرامج الضرورية لنظام التشغيل في النواة.

الآن أصبحت النواة صغيرة الحجم لا تحتاج إلى مساحة كبيرة بالذاكرة، لذلك سميت النواة المصغرة (small microkernel).



شكل رقم (3-12): نظام التشغيل الحديث

12.3.2.1. فوائد النواة المصغرة (microkernel)

استخدام النواة المصغرة جعل نظام التشغيل مرن ومن السهل تعديله واستبداله ، ذلك لأن الجزء الكبير من نظام التشغيل ينفذ في وضع المستخدم. بالتالي ليس هنالك حاجة لإعادة تشغيل النظام.

إمكانية تحويل جزء نظام التشغيل الموجود في وضع المستخدم الى أجهزة أخرى (portable).

استخدام النواة المصغرة أضفى على نظام التشغيل الأحادي صفة تمكنه من التوسع ليخدم عدة معالجات بسهولة.

12.3.2.2. عيوب النواة المصغرة (microkernel)

يكثر الاتصال بين أجزاء نظام التشغيل نتيجة لوجود برامج إدارة العتاد بمساحة المستخدم مما يقلل الكفاءة.

12.3.3. نظام التشغيل الموزع

Distributed Operating System (DOS)

هذا النوع يدير الموارد لتبدو وكأنها جهاز واحد كبير، فهو بذلك يوفر خاصية الشفافية التي تخفي جزئيات العتاد والبرامج وانتشارها في الشبكة.

ينقسم نظام التشغيل الموزع إلى نوعين بناءاً على العتاد الذي يديره:

- نظام تشغيل يدير معالجات متعددة (multiprocessors)
- نظام تشغيل يدير أجهزة حاسوب متعددة متجانسة (homogenous multicomputers).

شبكة الحاسوب قد تربط أجهزة حاسوب:

- متشابهة في العتاد والبرامج (متجانسة).
- مختلفة في العتاد و/أو في البرامج (متباينة).

طريقة إدارة العتاد في نظام التشغيل الموزع تشبه إلى حد كبير تلك التي في نظام التشغيل العادي، لذلك سندرس كيف يقوم نظام التشغيل العادي بذلك، ثم نوضح الفروق بين النوعين.

12.3.4.نظم التشغيل متعددة المعالجات

يختلف نظام التشغيل هنا عن سابقه في أنه يدير عدة معالجات وتوضع البيانات التي يحتاجها نظام التشغيل لإدارة العتاد في ذاكرة مشتركة. الجدول التالي يوضح مقارنة بين نظام التشغيل الأحادي ونظام التشغيل المتعدد.

نظام التشغيل أحادي المعالجات	نظام التشغيل متعدد المعالجات
يدير معالج واحد	يدير عدة معالجات
يستخدم المعالج ذاكرة واحدة	كل المعالجات تستخدم ذاكرة مشتركة
بيانات إدارة العتاد توجد في ذاكرته	بيانات إدارة العتاد في الذاكرة المشتركة
التعديل في البيانات يتم بواسطة معالج واحد	تعديل البيانات يتم بأكثر من معالج

ليس هنالك وصول متزامن	يوجد وصول متزامن للبيانات

شكل رقم (12-4): مقارنة بين نظام التشغيل الأحادي ونظام التشغيل متعدد المعالجات

الوصول المتزامن للبيانات في المعالجات المتعددة هو إمكانية تعديل البيانات بواسطة أكثر من معالج في وقت واحد مما ينتج عنه أخطاء، لذلك على نظام التشغيل هنا حماية الذاكرة من الوصول المتزامن لضمان توافق البيانات (consistency).

نظم التشغيل الأحادية والتي خصصت للحاسبات الشخصية ليس من السهل إدارتها لنظام ذي عدة معالجات، ذلك لأن برامجها صممت لتنفذ في معالج واحد وبالتالي لن تستفيد من العتاد متعدد المعالجات إذا استخدمت نظام تشغيل أحادي.

نظم التشغيل الحديثة صممت منذ البداية لتدعم عدة معالجات وبرامجها تنفذ بالتوازي في كل المعالجات.

<u>يهدف نظام التشغيل المتعدد المعالجات إلى الآتي:</u>

- رفع الكفاءة بدرجة عالية.
- إخفاء عدد المعالجات عن المستخدم والتطبيقات (توفير الشفافية).
- تحقيق الشفافية في هذا النظام سهلة إلى حدٍ ما ، ذلك لأن أجزاء التطبيق المختلفة تتصل ببعضها البعض بنفس الطريقة التي تتصل بها في تعدد المهام بنظم التشغيل الأحادية. أي ان التطبيق الذي يعمل في النظام الأحادي بطريقة تعدد المهام يمكن أن يعمل في نظام تشغيل متعدد المعالجات بنفس الطريقة.

ما الفرق الرئيسي بين تعدد المهام و تعدد المعالجات ؟

في تعدد المهام لا توجد موازاة حقيقية، فالذي ينفذ البرامج معالج واحد ولكنه يحاول أن يوزع زمنه بين البرامج لتبدو وكأنها تعمل بالتوازي بينما الحقيقة أن المعالج لا يستطيع تنفيذ أكثر من أمر واحد في نفس الوقت. بينما في تعدد المعالجات تتم موازاة حقيقية حيث يعمل كل برنامج على معالج منفصل (كل أمر يمكن أن ينفذ في معالج في نفس الوقت).

كل البيانات بهذا النظام متوفرة بالذاكرة المشتركة بالتالي تتم كل الاتصالات عبر هذه الذاكرة المشتركة.

12.3.5. مشكلة النظام المتعدد

بما أن الوصول للبيانات متاح لكل المعالجات، فالمشكلة الوحيدة هي حماية البيانات ضد الوصول المتزامن (simultaneous access).

12.3.6. طرق الحماية ضد الوصول المتزامن

هنالك طريقتان يمكن استخدامهما للحماية ضد الوصول المتزامن هما:

- طريقة السيمافور (semaphore)
- طريقة المراقب (monitors)

12.3.6.1. طريقة السيمافور

نعبر عن السيمافور بعدد صحيح يحتمل قيمتين :

- القيمة الأولى تعبر عن وضع "تحت" (down).
- القيمة الثانية هي وضع "فوق" (up).

دائما نختبر قيمة السيمافور قبل التنفيذ واعتمادا على القيمة نتخذ الإجراء اللازم.

<u>القيمة الأولى ("تحت"):</u>

إذا كانت قيمة السيمافور أكبر من صفر: ننقص قيمة السيمافور ونستمر في العملية.

إذا كانت القيمة صفر يتم إيقاف العملية (منعها من التنفيذ) process is blocked.

<u>القيمة الثانية ("فوق"):</u>

إذا وجدت عمليات موقوفة (محجوزة) نقوم بتحريرها لتواصل عملها.

إذا لم توجد عمليات موقوفة نزيد قيمة السيمافور.

العمليات التي تم تحريرها يمكنها مواصلة عملها بالتحويل من وضع "تحت".

أهم خاصية في نظام السيمافور هو أنه إذا بدأت عملية "تحت" أو عملية "فوق" لا يمكن لعملية أخرى الوصول إلى السيمافور حتى تكتمل العملية أو يتم توقيفها (تحجز).

البرمجة باستخدام السيمافور عرضة للأخطاء، لذلك يفضل استخدامها في حماية البيانات المشتركة فقط.

أمّا استخدامها في البرمجة فينتج عنه شفرات (أكواد) غير مهيكلة (مرتبة) (unstructured code) مثلها مثل استخدام الأمر goto في لغات البرمجة.

لذلك تفضل معظم نظم التشغيل الحديثة طريقة الأخرى للحماية غير طريقة السيمافور، هي طريقة المراقب (monitor).

12.3.6.2. طريقة المراقب (monitor)

المراقب يشبه إلى حد كبير الكائن في اللغات كائنيه التوجه، حيث يمثل وحدة برمجية تتألف من متغيرات وإجراءات (procedures).

المتغيرات تكون خاصة وممنوع الوصول إليها إلا عبر الإجراءات الموجودة معها في نفس الوحدة البرمجية (monitor) ، مثل عملية التغليف في اللغة كائنيه التوجه (encapsulation).

أهم خاصية في المراقب:

لا يمكن استدعاء أكثر من إجراء من إجراءات المراقب في اللحظة الواحدة .

هذه هي الخاصية الأساسية التي تمنع الوصول المتزامن للمتغيرات الموجودة بالمراقب.

مثال (1)

إذا قامت عملية ما باستدعاء إجراء من المراقب (دخلت إلى المراقب)، و حاولت عملية أخرى استدعاء إجراء من نفس المراقب في نفس الوقت، سيتم حجز العملية الثانية حتى تكتمل العملية الأولى (تخرج من المراقب).

لزيادة التوضيح دعنا نستخدم المثال البرمجي التالي.

مثال (2)

البرنامج التالي يوضح كيف يمكننا منع الوصول المتزامن للمتغير x.

```
Monitor  mn {
private:        int x=0;
public:         int value(){ return x;}
void incr() {x++;}
void decr(){x--;}
}
```

شكل رقم (5-12): برنامج المراقب mn

يوفر المراقب mn الحماية على المتغير x، وذلك بمنع الوصول إليه إلا عبر الدوال الموجودة بالمراقب وهي ()value ، ()incr ، ()decr.

<u>و بما أنه:</u>

- لا يمكن استدعاء أكثر من إجراء في اللحظة الواحدة.
- ولا يمكن الوصول للبيانات إلا بواسطة إجراءات (procedures) المراقب.

نكون قد حققنا عملية منع الوصول المتزامن للمتغير x.

مثال (3)

ماذا يحدث إذا استدعت عملية ما الإجراء ()decr، وكانت قيمة x هي صفر ؟

في هذه الحالة لابد من منع العملية من تنفيذ الإجراء decr() (حجزها).

لمثل هذه الأغراض يمكن أن يحتوي المراقب على متغيرات شرطية.

12.3.6.3. المتغيرات الشرطية:

هي عبارة عن متغيرات تحتمل قيمتين، إحداهما تمثل حالة الانتظار (wait) بينما تمثل القيمة الثانية حالة الإشارة (signal).

إذا كانت هنالك عملية ما، داخل المراقب و استدعت حالة الانتظار في المتغير الشرطي، سيؤدي هذا إلى حجزها (إيقافها)، في نفس الوقت إذا كانت هنالك عملية أخرى في حالة الانتظار يمكنها الآن دخول المراقب وبدء العمل.

أثناء عمل العملية الثانية التي هي داخل المراقب، يمكن السماح للعملية الأولى التي تم إيقافها من المواصلة، وذلك بإرسال إشارة عبر المتغير الشرطي الذي تنتظره العملية الأولى.

العملية التي ولدت الإشارة لابد من أن تترك المراقب، وذلك لتجنب وجود أكثر من عملية داخل المراقب (عدم التزامن).

يمكننا تعديل المراقب cr في المثال السابق لتوضيح كيفية عمل متغيرات شرطية في المراقب.

```
Monitor cr{
private:  int x=0;
int blocked_process=0;
Condition unblocked;
public:
        int value(){ return x;}
        void incr(){
```

```
                    if(blocked_process==0)
                          x++;
                    else    Signal(unblocked);}
void decr(){
            if (x==0)
                          Blocked_process++;
            Wait(unblocked);
Blocked_process--;
             }
else
x--;
   }
}
```

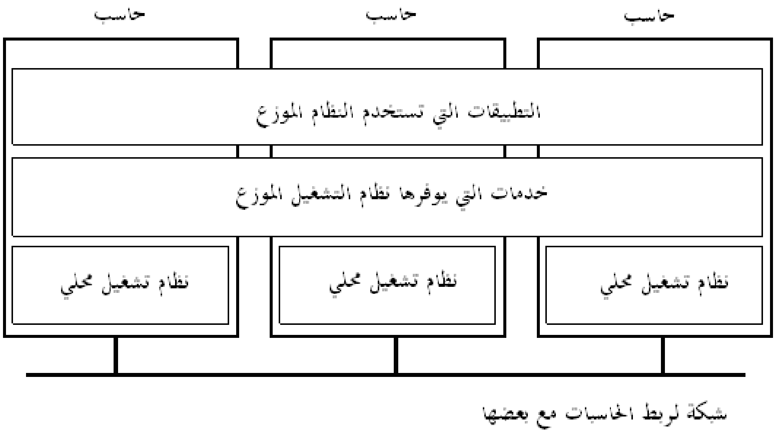
شكل رقم (12-6): برنامج المراقب crبعد إضافة متغيرات شرطية

12.3.6.4عيوب المراقب

عيبه انه مبني على لغة برمجة بعينها، مثلا يوجد بلغة جافا المراقب، ويعتبر كائن عادي يقوم بحماية بياناته من الوصول المتزامن عن طريق أوامر التزامن والانتظار (wait) والإشعار (notify) الموجودة بالكائنات. وقد تجد لغة برمجة أخرى تطبق المراقب بطريقة أخرى.

12.3.7. نظم التشغيل متعددة الحاسبات

نظم تشغيل الحاسبات المتعددة لديها هيكلية مختلفة عن نظم تشغيل المعالجات المتعددة. وهي أكثر تعقيداً منها، لأنه من الصعب وضع البيانات التي يحتاجها نظام التشغيل لإدارة الموارد في مكان مشترك. حيث لا توجد ذاكرة مشتركة هنا. لذلك يتم تبادل البيانات عن طريق الرسائل.



شكل رقم (12-7): نظام التشغيل الموزع

<u>كل جهاز لديه:</u>

نواة محلية خاصة به تحتوي على البرامج التي تدير الموارد المحلية لهذا الجهاز (مثل المعالج، الذاكرة ، القرص المحلي).

برنامج مخصص لمعالجة الاتصال بين العمليات حيث يتم من خلاله تبادل الرسائل مع الأجهزة الأخرى المرتبطة به.

هنالك طبقة برمجية مبنية ممتدة عبر نظم التشغيل المحلية تخفي تفاصيل الأجهزة وتباين نظمها لتبدو كأنها نظام واحد.

توفر هذه الطبقة الآتي:

- التنفيذ المتوازي و المتزامن للمهام المختلفة.
- ذاكرة افتراضية مشتركة تشبه تلك التي توجد في المعالجات المتعددة.

- تسهيلات أخرى مثل توزيع المهام على المعالجات.
- إخفاء أعطال العتاد ومعالجته.
- اخفاء تفاصيل تخزين البيانات
- إجراء الاتصال بين العمليات.
- أي أن كل ما تتوقعه في نظام التشغيل تجده في هذه الطبقة.

12.3.7.1. تبادل الرسائل (*message passing*)

بما أنه لا توجد ذاكرة مشتركة في نظام الحاسبات المتعددة ، فإن نظام التشغيل يوفر تبادل البيانات بين التطبيقات كوسيلة للتشارك في البيانات.

تختلف طريقة تبادل الرسائل باختلاف النظم.

12.3.7.2. الاتصال المعتمد للرسائل (reliable communication)

من المهم معرفة نوعية الاتصال بين الأجهزة هل هو معتمد (reliable) أو لا ؟

في الاتصال المعتمد نضمن وصول الرسائل للمستقبل. أما إذا كان الاتصال غير معتمدا فيجب على المرسل الانتظار حتى يأتيه إشعار من المستقبل بوصول الرسالة.

12.4. أنظمة الذاكرة المشتركة الموزعة (***DSM***)

كلمة DSM هي اختصار للكلمات distributed shared memory systems.

من خلال التجارب نجد أن برمجة الحاسبات المتعددة أصعب من برمجة المعالجات المتعددة. ففي نظام المعالجات المتعددة تكون عملية الوصول للبيانات المشتركة سهلة، ويمكن استخدام السيمافور أو المراقب لحماية الوصول المتزامن للبيانات. أما في نظام الحاسبات المتعددة فليس هناك غير طريقة واحدة للتشارك في البيانات هي تبادل الرسائل مما يعقد الموضوع ويربطه بالإتصال وهل هو معتمد أو لا.

هنالك بحوث كثيرة تحاول محاكاة الذاكرة المشتركة (emulating) في نظام تعدد الحاسبات.

الهدف من هذا المحاكي هو جعل التطبيقات تتعامل مع تعدد الحاسبات بنفس الطريقة التي تتعامل بها مع تعدد المعالجات ، حيث يوفر المحاكي هنا ذاكرة مشتركة خيالية (تشبه التي توجد في النظام متعدد المعالجات).

أحد طرق إنشاء الذاكرة الخيالية هي:

استخدام ذواكر كل الأجهزة لتكون ذاكرة خيالية كبيرة. تعمل هذه الذاكرة بنظام الصفحات حيث تقسم المساحات إلى صفحات (4 كيلو بايت أو 8 كيلو بايت). توجد هذه الصفحات حقيقة في ذواكر المعالجات (حيث لا توجد ذاكرة حقيقية مشتركة).

يتعامل المعالج مع المعلومات الموجودة بذاكرته بصورة عادية.

إذا احتاج المعالج لمعلومات غير موجودة في ذاكرته المحلية، يحدث الآتي:

- يرسل المعالج الذي يريد المعلومات رسالة لنظام التشغيل.
- يبحث نظام التشغيل عن الصفحة المطلوبة ويحضرها للمعالج.
- يحضر نظام التشغيل الصفحة التي تحتوي المعلومات المطلوبة دون أن يشعر المستخدم بأن هذه الصفحة احضرت من ذاكرة معالج آخر.

مثال

من الشكل (12-8)، إذا طلب المعالج 1 معلومات من الصفحات 0، 2، 5، 9، سيحصل عليها من ذاكرته المحلية. للحصول على معلومات من صفحات أخرى لابد من الاستعانة بنظام التشغيل، مثلا للحصول على معلومة من الصفحة 10 ، يرسل المعالج طلب لنظام التشغيل، الذي يقوم بدوره بنقل الصفحة 10 من المعالج 2 إلى المعالج 1، كما في الشكل (12-7).

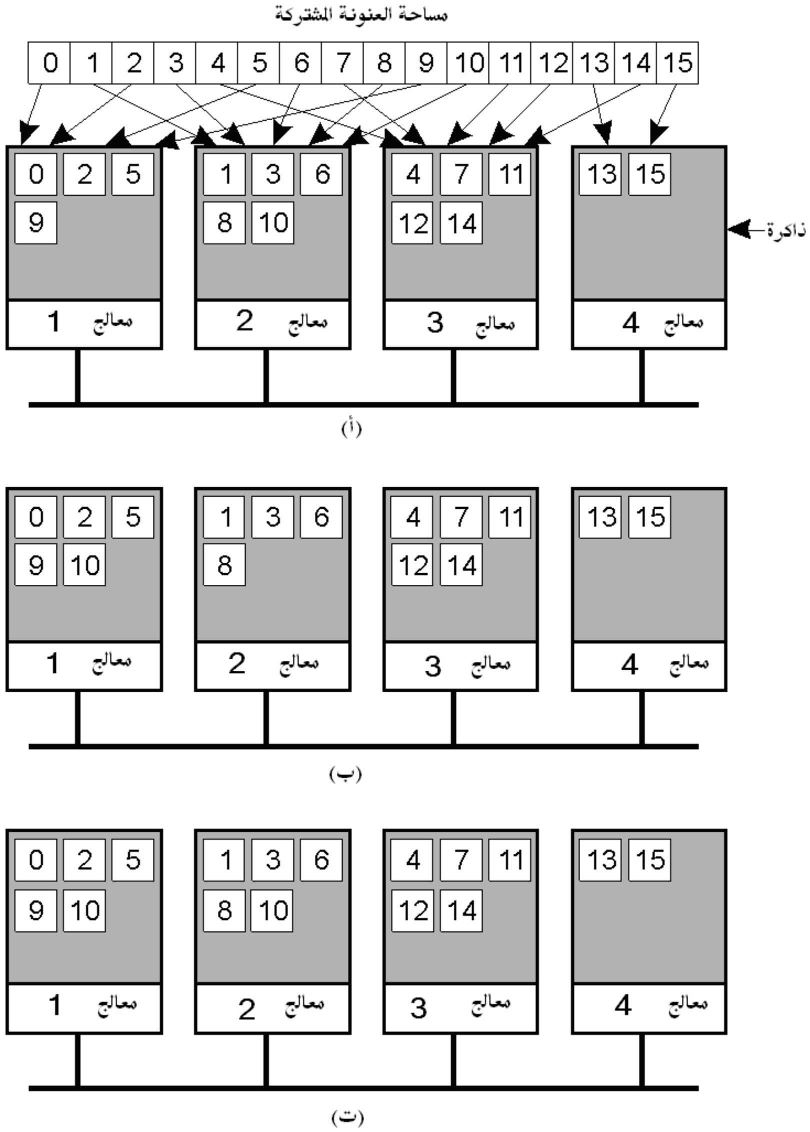
12.4.1. بعض التحسينات في نظام DSM

يمكننا تطوير نظام DSM لتحسين الأداء وذلك عن طريق تكرار صفحات القراءة فقط (read only).

مثال

إذا كانت الصفحة 10 في المثال السابق تحتوي معلومات للقراءة فقط وطلبها المعالج 1 فيمكن نسخها من المعالج 2 إلى المعالج 1 (بدلاً من نقلها كما في المثال السابق)، حيث يصبح لدينا نسختان من الصفحة 10، نسخة بالمعالج1 ونسخة ثانية بالمعالج 2، بهذه الطريقة نمكن المعالجين من استخدام نفس الصفحة بدون الحاجة إلى تبادلها بينهما.

الطريقة أعلاه يمكن تعميمها لتستخدم مع صفحات القراءة فقط والصفحات الأخرى أيضاً. إذا كانت الصفحة ليست للقراءة فقط ولكننا نستخدمها لقراءة معلومات منها و لا نحتاج حالياً للكتابة عليها، فيمكن معاملتها معاملة صفحات القراءة فقط وبالتالي يمكن نسخها وتكرارها على كل المعالجات التي تحتاجها.



شكل رقم (12-8)

المشكلة تكمن في إذا قام معالج بتعديل صفحة مكررة، هذا ينتج عنه عدم توافقية في المحتوى، أي أن الصفحات المتكررة قد تختلف محتوياتها بسبب تعديلات متباينة من قبل بعض المعالجات (inconsistency).

لذلك لابد من إجراء معين لضمان توافق البيانات لنفس الصفحة مع نسخها المكررة، فإذا قام معالج بتعديل بيانات على صفحة لابد من إجراء هذا التعديل على كل النسخ المكررة للصفحة.

يمكن أيضاً السماح بوجود نسخ مختلفة لنفس الصفحة حتى ولو قام كل معالج بتعديل نسخته مما ينتج عنه نسخ مختلفة (inconsistence). هذه الطريقة بالتجربة وجد أنها تزيد الكفاءة كثيراً. ولكنها تزيد من تعقيد البرمجة، حيث لابد من مراعاة عدم تطابق محتوي الصفحات حين كتابة البرامج.

تعقيد البرمجة التي تقود لها هذه الطريقة تلغي الهدف الرئيسي من بناء نظام DSM، حيث هدفه الأول هو تيسير البرمجة. لذلك تنفي هذه الطريقة هدف DSM الرئيسي.

12.4.2. تحديد حجم الصفحات يؤثر مباشرة في جودة نظام DSM.

كلفة إرسال صفحة عبر الشبكة يعتمد على إعدادات الإرسال (setting up transfer) وليس على كمية البيانات المرسلة. لذلك إذا كان حجم الصفحة كبير فهذا يقلل عدد مرات الإرسال. ولكن من جانب آخر إذا كانت لدينا صفحة تحتوي جزأين من البيانات وكل جزء يحتاجه برنامج في جهاز مختلف، على نظام التشغيل في هذه الحال إرسال هذه الصفحة باستمرار بين المعالجين. وجود بيانات في صفحة واحدة لبرنامجين مختلفتين يسمى المشاركة الخطأ (false sharing).

هنالك شد وجذب بين تحسين الأداء مقابل تعقيدات البرمجة في النظام متعدد الحاسبات. حيث نحصل على أداء جيد ولكن ببرمجة معقدة إذا استخدمنا طريقة تبادل الرسائل في النظام. أما إذا استخدمنا نظام DSM فستكون البرمجة سهلة ولكن الأداء ليس بالقدر المطلوب.

12.5. نظم تشغيل الشبكات ***Network Operating System (NOS)***

هذا النوع هو عبارة عن مجموعة من الأجهزة والموارد المنفصلة عن بعضها البعض، ولكل جهاز نظام تشغيله الخاص والذي يدير موارده. تتعاون هذه الأجهزة لتوفر خدماتها لبعضها البعض.

يتعامل هذا النوع من نظم التشغيل مع الحاسبات المتعددة المتباينة (Heterogeneous Multicomputers).

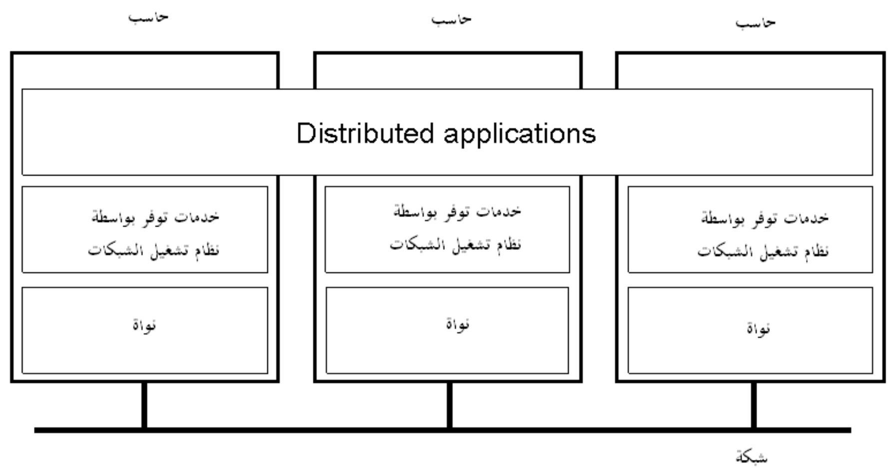
يعمل نظام تشغيل الشبكات بنفس الطريقة التي يعمل بها نظام التشغيل العادي (uniprocessor)، ولكن يختلف عنه في كونه يتيح خدمات الأجهزة المحلية (local services) للأجهزة البعيدة (remote clients).

تفتقر نظم تشغيل الشبكات إلى العديد من الخصائص والخدمات التي يحتاجها النظام الموزع مثل خاصية الشفافية (transparency).

يمكن إضافة هذه الخصائص التي لا توجد بنظم تشغيل الشبكات والتي تمنعه من دعم عمل النظم الموزعة، وذلك عن طريق جمعها في طبقة برمجية تسمى الطبقة الوسيطة (middleware) ووضعها على نظام تشغيل الشبكات، بهذه الطريقة يصبح النظام قادراً على توفير بيئة مناسبة للنظم الموزعة.

مقارنة بنظام التشغيل الموزع (DOS)، لا يفترض نظام تشغيل الشبكات وجود عتاد متجانس ، ولا يدير النظام كوحدة متماسكة. وإنما هو عبارة عن مجموعة من الحاسبات الأحادية والمستقلة عن بعضها البعض والتي قد تكون مختلفة في كل شيء حتى في نظام التشغيل. هذه الأجهزة متصلة بعضها البعض عن طريق رابط ما.

يوفر نظام تشغل الشبكات تسهيلات تمكن المستخدمين من الاستفادة من الخدمات المتاحة في هذه الأجهزة.



شكل رقم (12-9)

يعمل نظام تشغيل الشبكات مع أجهزة متباينة ولكن هذا لا ينفي عمله مع أجهزة متجانسة في العتاد و مستقلة عن بعضها البعض ، و مرتبطة مع بعضها البعض برابط معين.

12.5.1. الخدمات التي يوفرها نظام تشغيل الشبكات

يمكننا وصف نظام تشغيل الشبكات من خلال الخدمات التي يوفرها للتطبيقات والمستخدمين. من الخدمات التي نجدها في معظم نظم تشغيل الشبكات الآتي:

خدمة الدخول على جهاز بعيد والتعامل معه كأنه جزء من جهازك المحلي. حيث وبعد الدخول على الجهاز البعيد ترسل أوامرك عبر الشبكة لذاك الجهاز وترى النتائج على جهازك المحلي.

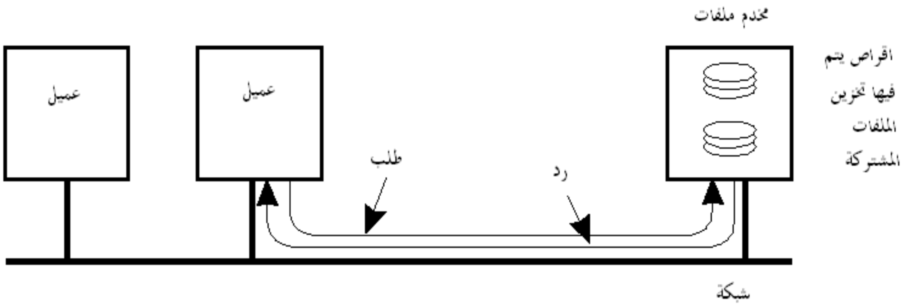
يمكن ان تدخل على أكثر من جهاز بعيد في نفس الوقت وتتعامل مع هذه الأجهزة البعيدة كأنها جزء من جهازك.

خدمة نسخ ملفات من أجهزة بعيدة، مثلا الأمر :

Rcp m1:f1   m2:f2

يقوم بنسخ الملف f1 من الجهاز m1 الى الجهاز m2 وبالاسم f2.

خدمة نظام الملفات حيث يخصص جهاز خادم (server) تخزن به الملفات وتتاح لكل الأجهزة. يقوم برنامج نظام الملفات (file server) الموجود بهذا الخادم باستلام طلبات المستخدمين من الأجهزة العميلة (clients)، التي قد تكون لقراءة ملف أو تخزين ملف أو غيره. حيث تنفذ الطلبات بواسطة نظام الملفات ويرسل الرد للأجهزة العميلة، الشكل (12-10).



شكل رقم (12-10)

نظام تشغيل الشبكات هو أكثر بدائية من نظام التشغيل الموزع، الفرق بين الاثنين هو أن الأخير يحاول صبغ النظام بالشفافية (transparency).

12.5.2. عيوب نظام تشغيل الشبكات

عدم وجود شفافية (transparency) في نظام تشغيل الشبكات أوجدت به عيوب منها:

صعوبة الاستخدام، فمثلا عملية الدخول على جهاز بعيد أو نقل الملفات تحتاج إلى خبرة من المستخدم.

مشاكل إدارية مثل:

- استقلالية الأجهزة ونظم التشغيل في الشبكة تحتاج أحيانا إلى إدارة مستقلة.
- أحيانا تجد صعوبة في الدخول على جهاز بعيد ما لم يكن لديك حساب (account) في ذاك الجهاز.
- إذا أراد مستخدم التعامل بكلمة مرور واحدة فلابد من تغييرها في كل جهاز.
- أيضا قد يحتاج المستخدم معالجة أذونات الوصول (access permissions) في كل جهاز.
- ليس هنالك طريقة بسيطة لتغيير الأذونات لأنها متشابهة في كل مكان.

- هذه الطريقة اللامركزية للحماية (security) تجعل من الصعب أحياناً حماية نظام تشغيل الشبكات من الهجمات التخريبية.

12.5.3. مزايا نظم تشغيل الشبكات

يعتبر نظام تشغيل الشبكات سهل التعديل مقارنة مع نظام التشغيل الموزع. فيمكننا إضافة محطة أو إزالة محطة من النظام بسهولة، وأحيانا يتم ذلك فقط بتوصيل المحطة مع كيبل الشبكة و إبلاغ بقية المحطات بوجودها.

12.6. الطبقة الوسيطة ***middleware***

ليكون النظام الموزع حقيقياً لابد من توفر نقطتين مهمتين فيه:

- حاسبات مستقلة (independent computers).
- نظام واحد متماسك (single coherent system).

الأولى نجدها متوفرة في نظام تشغيل الشبكات ويفتقر للثانية، بينما تتوفر الثانية في نظام التشغيل الموزع ولا توجد به الأولى. فكلا النظامين لا يوفر النظام الموزع الحقيقي.

<u>إذن كيف الوصول إلى النظام الموزع الحقيقي ؟</u>

تحدثنا مسبقاً عن صفات الأنظمة الموزعة وأهمها:

- التوسعية.
- الانفتاحية.
- الشفافية.
- سهولة الاستخدام (هذه الخاصية مطلوبة في كل النظم موزعة وغيرها).

هذه الصفات إذا تمكنا من توفيرها في نظام فسيصبح تقريبا نظاماً موزعاً حقيقياً. إذا نظرنا إلى نظام تشغيل الشبكات نجد أن به خاصية التوسعية وخاصية الانفتاحية، أما

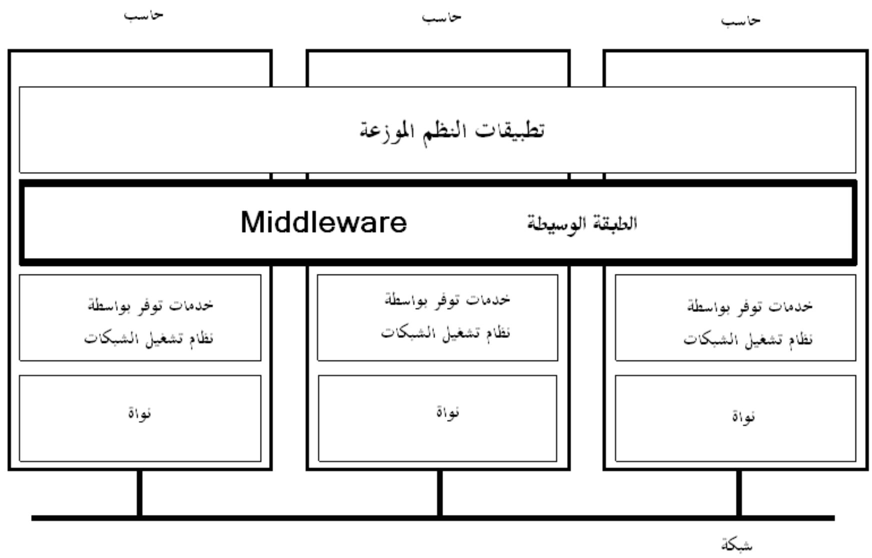
نظام التشغيل الموزع فهو سهل الاستخدام و عالي الشفافية (transparence). فكلا النظامين ناقص. فما هو الحل؟

الحل يكمن في إضافة طبقة برمجية (layer) لنظام تشغيل الشبكات تضيف إليه الخواص الناقصة مثل سهولة الاستخدام والشفافية.

12.6.1. مكان الطبقة الوسيطة positioning middleware

كثير من التطبيقات الموزعة تستخدم الواجهات البرمجية التي يوفرها نظام تشغيل الشبكات. مثلا قد تحتاج إلى إجراء اتصال باستخدام الـ sockets التي تسمح للعديد من العمليات بأجهزة مختلفة لتبادل الرسائل.

قد تستخدم التطبيقات بعض واجهات (interfaces) نظام الملفات المحلى. نجد هنا أنه من الصعب إخفاء تفاصيل التوزيع (distribution). لذلك الحل هو وضع طبقة بين نظام تشغيل الشبكات والتطبيقات، هذه الطبقة تسمى الطبقة الوسيطة (middleware) وتوضع في الوسط بين نظام تشغيل الشبكات والتطبيقات.



شكل رقم (12-11): الطبقة الوسيطة

كل نظام محلي (هو جزء من نظام تشغيل الشبكات) يفترض أن يوفر إدارة الموارد المحلية بالإضافة إلى توفير أيسر طرق الاتصال التي تربطه بالأجهزة الأخرى، هذا يعني أن الطبقة الوسيطة نفسها لا تدير الأجهزة منفردة وإنما يترك هذا لنظام التشغيل المحلي.

الهدف المهم هو إخفاء التباين الموجود بالنظام عن التطبيقات.

العديد من نماذج الطبقة الوسيطة توفر مجموعة كاملة من الخدمات وهي تشجع استخدامات الواجهات أكثر من أي شي آخر.

12.6.2. المشاكل

الاتفاق على طبقة وسيطة موحدة ليس سهلاً. لذلك لابد من منظمة تُعرِّف وتضع المعايير والأسس التي تبني طبقة وسيطة معيارية. هنالك العديد من المعايير التي وضعت ولكنها ليست متوافقة مع بعضها البعض، والأسوأ من ذلك أن المنتجات التي تطبق نفس المعايير (لكن من مصانع مختلفة) من الصعب أن تتعامل مع بعضها البعض.

12.6.3. نماذج الطبقة الوسيطة (middleware models)

لتبسيط عملية تطوير التطبيقات الموزعة وجعل تكاملها أيسر ما يكون، تضع معظم الطبقات الوسيطة نموذج لوصف التوزيع والاتصال.

من أبسط النماذج الموجودة التعامل مع كل شيء على أنه ملف. هذه الطريقة كانت أولاً في ينكس (UNIX).

12.6.4. أمثلة لنماذج طبقات وسيطة

12.6.4.1. نظام الملفات الموزع

هو عبارة عن طبقة تضيف نوع خدمات واحد لنظام تشغيل الشبكات حيث يتم دعم شفافية توزيع الملفات. مما ساهم في انتشار هذه الطبقة هو قابليتها للتوسع. يتعامل المستخدم مع الملفات الموزعة كأنها في مكان واحد.

12.6.4.2. نداء الإجراء البعيد (*RPC*)

يمكن لبرنامج في جهاز نداء إجراء في برنامج آخر بجهاز آخر (بعيد)، وكأن ذاك الإجراء موجود بالجهاز المحلي، فالبرنامج الذي قام بنداء الإجراء لا يميز ما إذا كان الإجراء بعيداً أو قريباً وما إذا كان هنالك اتصال تم أو لا، الفرق الوحيد الذي يحس به المستخدم هو أن تنفيذ الإجراء البعيد يستغرق وقتاً أطول عن نداء الإجراء العادي (ذلك للاتصالات التي تتم والرسائل التي تتبادل).

12.6.4.3. نداء الكائن البعيد (*distributed objects invocations*)

هي تطبيق لفكرة نداء الإجراء البعيد أعلاه، لكن على الكائنات (objects)، فالطريقة (method) البعيدة المستدعاة توجد في كائن بالجهاز البعيد. وهي تعمل بنفس الطريقة حيث لا يميز البرنامج الذي استدعى الطريقة (method) هل هي بعيدة أم بالجهاز نفسه (شفافية النداء).

12.6.4.4. نموذج الوثائق الموزعة (*distributed documents*)

هذا النموذج هو وراء نجاح الويب. ففي الويب تنظم المعلومات في شكل وثائق حيث تخزن كل الوثائق في الأجهزة بشفافية (لا نعرف موقع الوثيقة الحقيقي).

هذه الوثائق متنوعة المحتوى فقد يكون فيها نصوص، أصوات، فيديو، صور، وغيرها. ونتعامل مع الوثيقة للوصول إليها عبر عنوان URL.

مثال

لفتح الموقع http://www.sustech.edu نكتبه في شريط العنوان بالمتصفح، ونتعامل مع هذا العنوان فقط لنحصل على الوثيقة، فلا نحتاج معرفة معلومات إضافية عن الوثيقة مثل مكان تخزينها :

- في أي بلد.
- في أي جهاز.
- ما نظام تشغيل الخادم.
- ما اللغة المكتوب بها الموقع.

- ما نوع عتاد الخادم.
- ما شكل الشبكات التي تربطني به.

وإذا تم نقل هذا الموقع من خادم ويب إلى خادم ويب آخر، ببلدة ثانية، بدولة أخرى، بقارة أخرى، بنظام تشغيل مختلف، فسأظل أصل إليه بنفس الطريقة (شفافية المكان).

12.6.5. خدمات الطبقة الوسيطة (middleware services)

هنالك العديد من الخدمات التي توفرها الطبقة الوسيطة نذكر منها:

- خدمة الشفافية (إخفاء التفاصيل): وذلك بإخفاء تفاصيل الاتصال بين العمليات وتبادل الرسائل في الشبكة. مثل ما توفره طريقة نداء الإجراء البعيد من الوصول للدوال البعيدة بشفافية (RPC).
- وما يوفره نظام الملفات الموزع (distributed file systems) من الوصول للبيانات البعيدة بشفافية. فيستطيع المستخدم الوصول لملفات موزعة في أجهزة مختلفة وكأنها في مكان واحد، ولا يحتاج تذكر أماكن هذه الملفات ولا في أي أجهزة تتوزع.
- خدمة التسمية (naming): تستخدم لإخفاء تفاصيل المورد والوصول إليه بسهولة مثلما يحدث في الويب، حيث يمثل العنوان (URL) الإسم الذي يشير إلى مكان الوثيقة بشفافية.
- الحماية (security): من الخدمات المهمة التي يجب أن تتوافر بالطبقة الوسيطة الحماية.

12.6.6. انفتاح الطبقة الوسيطة (openness)

بما أن النظم الموزعة الحديثة تكون في شكل طبقة وسيطة تُبنى على نظام تشغيل الشبكات، فالتطبيقات الموزعة سوف تُبنى على الطبقة الوسيطة وبالتالي ليس هنالك علاقة مباشرة مع نظام تشغيل الشبكات والنظام الموزع. أي أن التطبيقات تكون مستقلة عن نظام تشغيل الشبكات ولكنها بالمقابل مرتبطة بالطبقة الوسيطة ومعتمدة عليها. وهذا يولد مشكلة هنا أن الطبقة الوسيطة تكون غير منفتحة بالشكل المطلوب.

تحدد درجة انفتاح النظام بالواجهات (interface) التي يوفرها. فلا بد أن تكون الواجهات كاملة لتفي بكل متطلبات المستخدم.

عدم توافر واجهات كاملة يدفع بمطوري الأنظمة لإضافة واجهاتهم الخاصة لإكمال النقص الموجود في واجهات الطبقة الوسيطة. هذا يعني أن كل مجموعة مصممي نظم تضيف واجهاتها، مما يقود إلى تباين الواجهات المضافة بالتالي تصبح تطبيقات كل جهة معتمدة على واجهاتها التي أضافتها للطبقة، وهذا يؤدي إلى انتفاء خاصية التنقلية (portability) بين تطبيقات الطبقة الوسيطة الواحدة.

12.7. تمارين

- اذكر ثلاثة أمثلة لنماذج طبقة وسيطة ؟
- ما الفرق بين نظام التشغيل الموزع، نظام تشغيل الشبكات، ونظام التشغيل العادي؟
- ما المهمة الأساسية للطبقة الوسيطة؟
- هل تعتقد أن نظام التشغيل الموزع كافٍ لتوفير بيئة ملائمة للأنظمة الموزعة ؟ و لماذا ؟
- ما هي مهام برمجيات النظم الموزعة ؟

الملاحق

ملحق (أ)

المراجع

Tanenbaum, A.S., (2009), Modern operating system, 3rd edition, Pearson Education (Prentice Hall)	1
Galvin, S. (2009), OPERATING SYSTEM CONCEPTS, 8th Edition, Addison Wesley	2
Que Corporation (1990), USING UNIX, Que	3
Flynn, I.M., McHoes, A.M. (2001), Understanding operating systems (3rd ed.), Brooks/Cole Publishing Co. Pacific Grove, CA, USA	4
Crowley, C. (1997), Operating Systems A Design-Oriented Approach, IRWIN.	5

ملحق (ب)

المصطلحات

Process	عملية
Thread	الخيط
Scheduling	الجدولة
Deadlock	الإختناق
Memory	الذاكرة
Virtual memory	الذاكرة الظاهرية / الافتراضية
Performance	الأداء
Efficiency	الكفاءة
Cluster	تجمع
Multiprocessor	حاسب متعدد المعالجات
Multitasking	تعدد المهام
Cache memory	الذاكرة المخبأة
Main memory	الذاكرة الرئيسية
Registers	المسجلات
Processor	المعالج
Resource	المورد
Page Fault	خطأ الصفحة

Page Replacement	استبدال الصحفات
Swap	التبديل
Main memory	الذاكرة الرئيسية
Cache memory	الذاكرة المخبأة
Device drivers	التعريفات/سواقات الأجهزة

ملحق (ج)

أجزاء الثانية

اسم القياس	جزء من الثانية
yoctosecond [ys]	0,000 000 000 000 000 000 000 001
zeptosecond [zs]	0,000 000 000 000 000 000 001
attosecond [as]	0,000 000 000 000 000 001
femtosecond [fs]	0,000 000 000 000 001
picosecond [ps]	0,000 000 000 001 [trillionth]
nanosecond [ns]	0,000 000 001 [billionth]
microsecond [μs]	0,000 001 [millionth]
millisecond [ms]	0,001 [thousandth]
centisecond [cs]	0.01 [hundredth]
second [s]	1.0
	مقارنة بين الثواني والسنين
= 1 minute [mean solar]	60 seconds
= 1 minute [sidereal]	59.83617 seconds
= 1 hour	60 minutes

24 hours	= 1 day
3,600 seconds	= 1 hour
86,400 seconds	= 1 day [mean solar]
86,164.09 seconds	= 1 day [sidereal]
7 days	= 1 week
168 hours	= 1 week
14 days	= 1 fortnight
28, 29, 30 or 31 days	= 1 month
365 days	= 1 year
366 days	= 1 leap year
12 months	= 1 year
31,536,000 seconds	= 1 year [calendar]
31,558,150 seconds	= 1 year [sidereal]
31,556,930 seconds	= 1 year [tropical]
9,460,550,000,000,000 metres	= 1 light year
299,792,458 metres per second (m/s)	= speed of light
Parsec	= approx. 3.25 light years

من الموقع: http://www.simetric.co.uk/si_time.htm

ملحق (د)

التحكم في العمليات على لينكس (توزيعة أوبونتو)

في هذا الملحق نسرد بعض البرامج التي تستخدم استدعاءات النظام في لينكس للتحكم في العمليات من داخل برامج C.

خطوات تنفيذ البرنامج:

قبل البدء في سرد أمثلة البرامج نوضح هنا خطوات كتابة وترجمة وتنفيذ برامج C على لينكس (توزيعة أوبونتو):

- أفتح الطرفية
- أكتب الأمر gedit (محرر لكتابة شفرة البرنامج).
- أكتب البرنامج أدناه في المحرر.
- أحفظ الملف بالاسم creatProcess.c
- ترجم البرنامج بالأمر:
  - cc -o creatProcess   creatProcess.c
- نفذ البرنامج بعد الترجمة وتصحيح الأخطاء بالأمر:
  - ./ creatProcess

برنامج (1)

```
#include <stdio.h>

#include <sys/types.h>

#define MAX_COUNT 200

void ChildProcess(void); /* child process prototype */

void ParentProcess(void); /* parent process prototype */

void main(void) {

        pid_t pid;
```

```
        pid = fork();
        if (pid == 0)
        ChildProcess();
        else ParentProcess();
 }
void ChildProcess(void) {
        int i=5;
        printf(" This line is from child, value = %d\n", i);
        printf(" *** Child process is done ***\n");
 }
void ParentProcess(void) {
         int i=9;
        printf("This line is from parent, value = %d\n", i);
        printf("*** Parent is done ***\n");
}
```

جرب البرنامج أعلاه ووضح ما هو المخرج ؟

<u>برنامج (2)</u>

البرنامج التالي يوضح كيفية استخدام استدعاءات النظام بواسطة برنامج C++، حيث تم لإنشاء عملية باستدعاء fork()، ثم اختبار القيمة الراجعة منها، للتميز بين العملية الإبن والعليمة الأب.

```
#include <iostream>
#include <string>

// Required by for routine
#include <sys/types.h>
#include <unistd.h>

using namespace std;

int globalVariable = 2;

main()
{
   string sIdentifier;
   int    iStackVariable = 20;

   pid_t pID = fork();
   if (pID == 0)                // child
   {
      // Code only executed by child process

      sIdentifier = "Child Process: ";
      globalVariable++;
      iStackVariable++;
    }
    else if (pID < 0)            // failed to fork
    {
        cerr << "Failed to fork" << endl;
        exit(1);
        // Throw exception
    }
    else                                  // parent
    {
      // Code only executed by parent process

      sIdentifier = "Parent Process:";
    }

    // Code executed by both parent and child.

    cout << sIdentifier;
    cout << " Global variable: " << globalVariable;
```

```
    cout << " Stack variable: "  << iStackVariable << endl;
}
```

البرنامج (3)

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main() {
pid_t pid;  /*  رقم العملية  */

char *message;
int n;

printf("fork program starting");
استدعاء الأمر الذي ينشي (يفرخ) عملية جديدة ويخزن رقم العملية في متغير
pid = fork();

switch(pid) {
   case -1: exit(1);
   case 0: message = "this is the child process"; n = 3;

              break;
   default: message = "this is the parent process"; n = 6;

              break;

   }
for(; n > 0; n-) {
   puts(message);
   sleep(1);

    }
exit(0);
```

```
}
```

شرح البرنامج

عندما نستدعى النظام بالامر fork ، فهذا يعني أننا نريد من نظام التشغيل إنشاء عملية، سيتم تنفيذ الأمر بواسطة نظام التشغيل وترجع قيمة من بعد تنفيذ الأمر:

```
pid = fork();
```

تخزن القيمة الراجعة بالمتغير pid، الذي يعتبر رقم تعريف العملية (وهو رقم غير متكرر، فكل عملية تنشأ سيكون ليدها رقم تعريف فريد (unique)) يسمى رقم تعريف العملية pid.

إذا كان هنالك خطأ في التنفيذ ستكون القيمة الراجعة 1-.

أيضا يمكننا إنهاء عملية باستخدام الأمر:

```
kill -9 [PID] $
```

حيث يمثل PID رقم العملية المراد إنهاءها، مثلا لإنهاء العملية رقم 1337 سننفذ الأمر التالي على الطرفية:

```
$ $kill -9 1337
```

أيضا يمكننا إنهاء نفس العملية من داخل برنامج C بالأمر:

```
kill( 1337, SIGKILL );
```

برنامج (4)

في البرنامج التالي نقوم بإنشاء عملية بالأمر fork()، ثم تتفذ العملية بالامر execlp أو exec أو غيرها من دوال التنفيذ. كما يمكن إنهاء العملية بالأمر exit(). أيضا الأمر wait يستخدم لجعل عملية تنتظر عملية أخرى مثلا.

```
int main()

{
```

```
    pid_t  pid;
    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait (NULL);
        printf ("Child Complete");
        exit(0);
    }
}
```

برنامج (5)

```
#include <stdio.h>

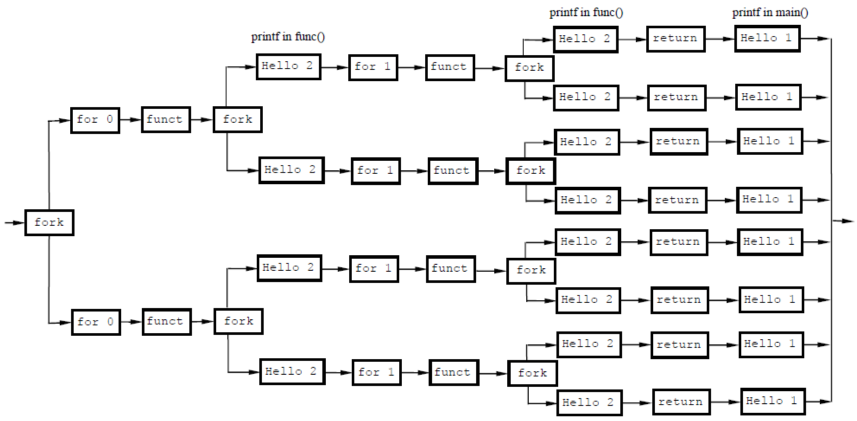
void funct ()
 {
  fork ();
  printf ("Hello 2 !\n");
  return;
 }

 int main ()
  {
   int i;
   fork ();
   for (i = 0; i < 2; i++)
     funct ();
   printf ("Hello 1 !\n");
   exit (0);
  }
```

مخرج البرنامج أعلاه سيكون كما يلي:

```
Hello 2 !
Hello 2 !
Hello 1 !
Hello 2 !
Hello 2 !
Hello 1 !
Hello 2 !
Hello 2 !
Hello 1 !
Hello 2 !
Hello 1 !
Hello 2 !
Hello 2 !
Hello 1 !
Hello 2 !
Hello 1 !
Hello 2 !
Hello 1 !
Hello 2 !
Hello 1 !
```

نجد عبارة Hello 2 تظهر 12 مرة، و Hello 1 تظهر 8 مرات. ذلك لأن fork في الدالة funct() تولد نسختين من التكرار for. ترتيب هذه العبارات غير محدد وقد يختلف من مرة لأخرى. الرسم التالي يوضح خطوات التنفيذ.



برنامج (6)

أكتب برنامج بلغة C يمثل العملية الأب، يقوم بإنشاء عملية إبن سميها child1 باستدعاء fork(). والعملية الإبن بدورها تنشئ عملية أبن اسمها child2. العمليات الأبناء هي صور من العملية الأب. بينما تقوم العملية الأب بحساب مربع العدد 3، والعملية child1 تحسب مربع العدد 4 بالتزامن مع العملية الأب، والعملية child2 تقوم بحساب مربع العدد 5. كل العمليات تظهر نتائجها في الشاشة مباشرة بعد عملية الحساب. تأكد من إنهاء العمليات بطريقة سليمة.

```
int pid = fork();
if (pid==0){
        pid=fork();
        if (pid==0) {
            printf(child 2 %d \n", 3*3);
            exit()
```

```
            }
            printf("child2 %d\n", 4*4);
            exit();
            }
Printf("parent %d\n",  5*5);
Wait(NULL);}
```

Wait(): توقف تنفيذ العملية الحالية حتى تنتهي العملية الإبن.

برنامج (6)

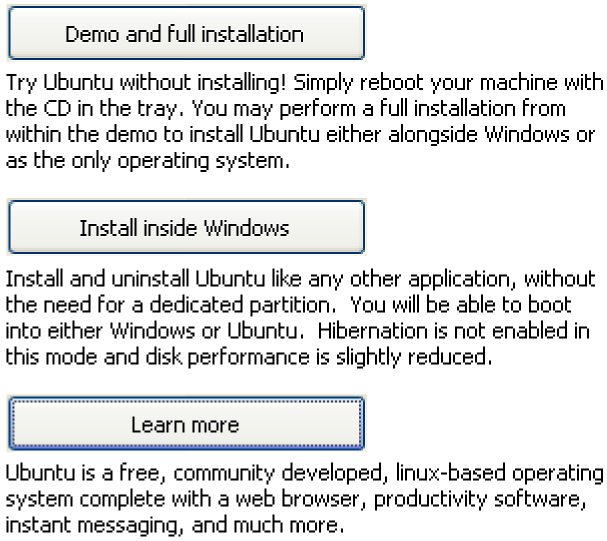
ما هو مخرج البرنامج التالي:

```
int pid=fork()
if (pid==0){
        while(1)
            printf("child…");
} else {
        while(1)
            printf("Parent …");}
```

ملحق (هـ)

تثبيت أوبونتو النسخة 8.10 – أو 9.04

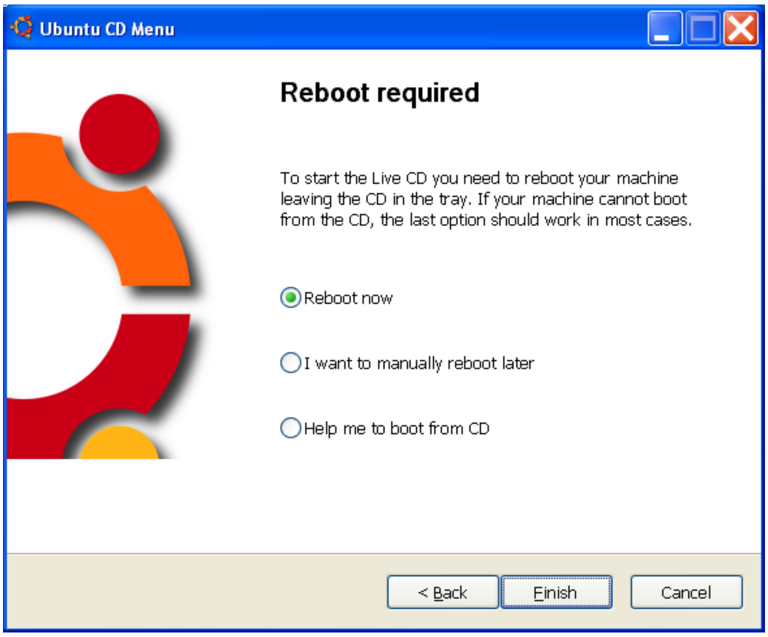
عند تشغيل اسطوانة أوبونتو لأول مرة تظهر الشاشة التالية :



- الخيار الأول: (Demo and full Installation) إذا كنت تريد تثبيت أوبونتو تثبيتا كاملا (full installation)، أو إذا أردت تجربته دون تثبيته في جهازك (live CD).
- الخيار الثاني: (Install inside windows) إذا كنت تريد تثبيت أوبونتو داخل ويندوز مثله مثل تثبيت أي ملف.
- الخيار الثالث: (Learn more) لمعرفة معلومات أكثر (عن طريق موقع أوبونتو).

الخيار الأول (الثبيت الكامل أو التجربة دون تثبيت)

إذا نقرنا على زر Demo and full Installation سيعمل النظام وبعدها ستظهر الشاشة التالية:



أختار Reboot now ثم انقر على Finish. سيتم إعادة تشغيل الكمبيوتر، ويبدأ أوبونتو في العمل. أو ببساطة ضع أسطوانة أوبونتو في السواقة ثم قم بإعادة تشغيل الكمبيوتر يدويا. بعد إعادة التشغيل ستظهر خيارات هي :

- Try Ubuntu without any change to your computer (تجربة أوبونتو دون أن تثبته في جهازك)
- Install Ubuntu (تثبت أوبونتو كاملا على جهازك)

تجربة أوبونتو (live CD)

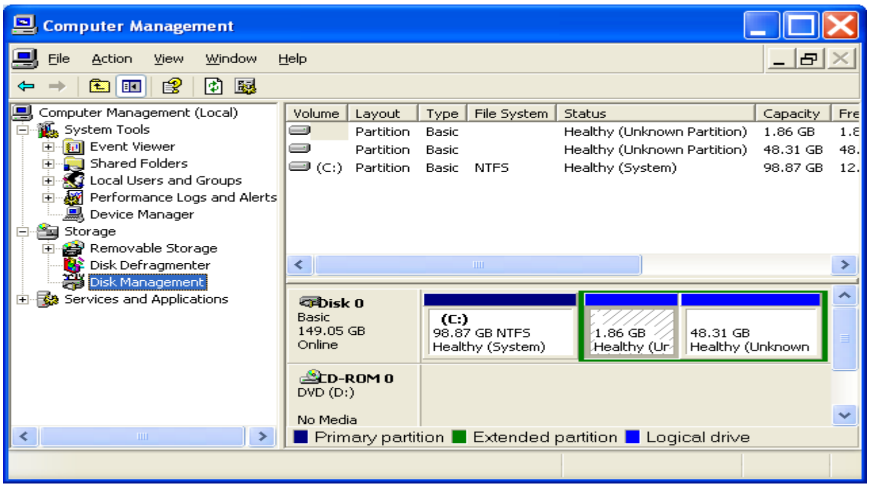
إذا اخترت (Try Ubuntu) : فهذا يعني أنك تريد تشغيل أوبونتو من الاسطوانة (live CD) دون أن يثبت على جهازك ودون أن يؤثر على جهازك بشي.

تثبيت أوبونتو النسخة الكاملة

أما إذا اخترت (Install Ubuntu) فهذا الأمر سيقوم بتثبيت أوبونتو على القرص الصلب لديك واستخدامه بصورة دائمة.

تأكد قبل التثبيت

من وجود قسم (partition) فارغ لتثبيت أوبونتو عليه (للتأكد من ذلك أفتح لوحة التحكم – أدوات إدارية -إدارة الحاسب (computer management) ثم إدارة الأقراص وتأكد من وجود قسم غير C للتثبيت.



أو عليك إنشاء قسم جديد في القرص الصلب. في حالة وجود قرص صلب غير مقسم (C فقط)، الأسهل أن تستخدم الطريقة الثانية في تثبيت أوبونتو والسهلة (تثبيت أوبونتو كملف في C دون الحاجة إلى تقسيم القرص الصلب وغلبته).

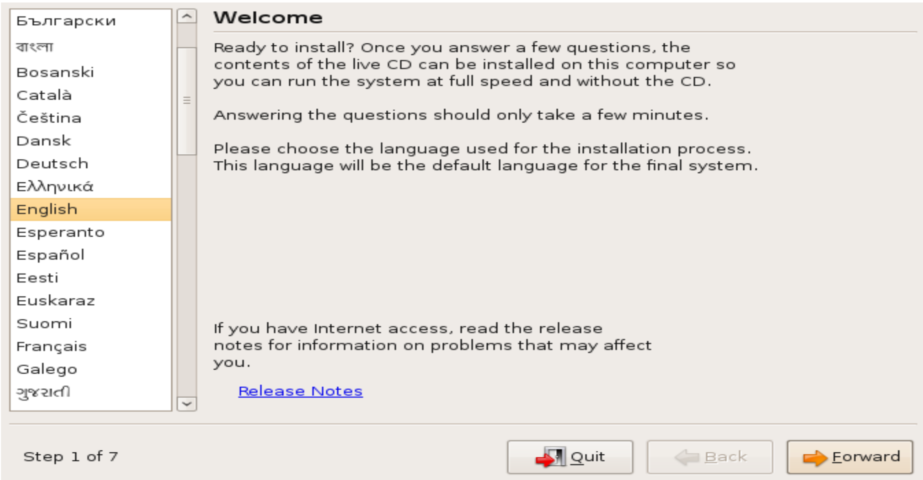
أما إذا كنت محترف وتريد تثبيت أوبونتو في قسم جديد (new partition)، فعليك استخدام برامج تقسيم الأقراص الصلبة الاحترافية مثل Acronis PartitionExpert أو Partition magic.

إذا لديك قسم آخر غير C مثلا D أنقل بياناتك منه إلى C وأتركه فارغا (لأنك ستفقد محتواه بعد تثبيت أوبونتو فيه.

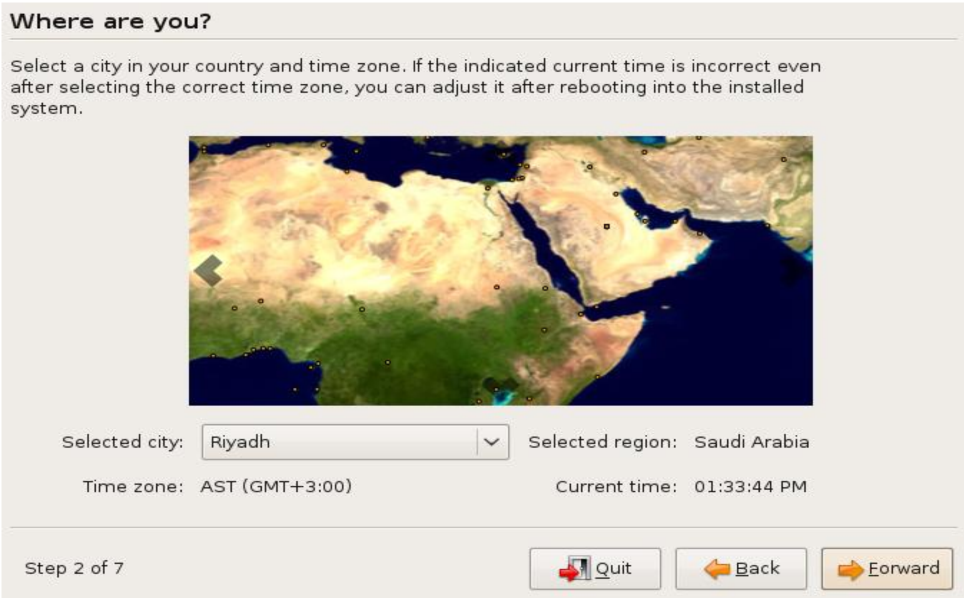
بداية التثبيت الكامل

إذا اخترت Install Ubuntu ، سيبدأ أوبونتو في العمل من الاسطوانة (قد يستغرق بعض الوقت)، ثم تظهر نافذة التثبيت، التي تتكون من سبع خطوات هي:

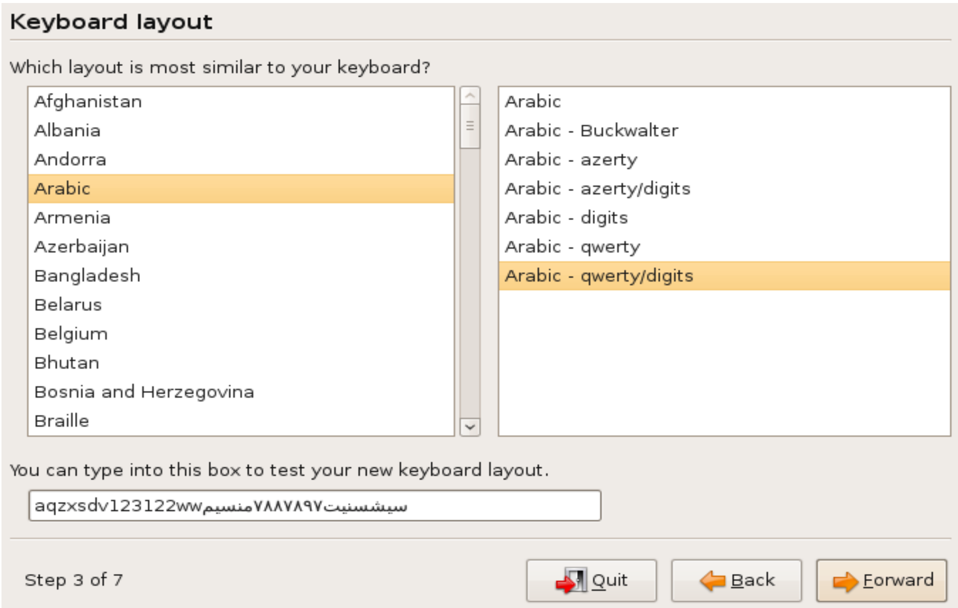
1. أختار لغة واجهة النظام (English) مثلا.



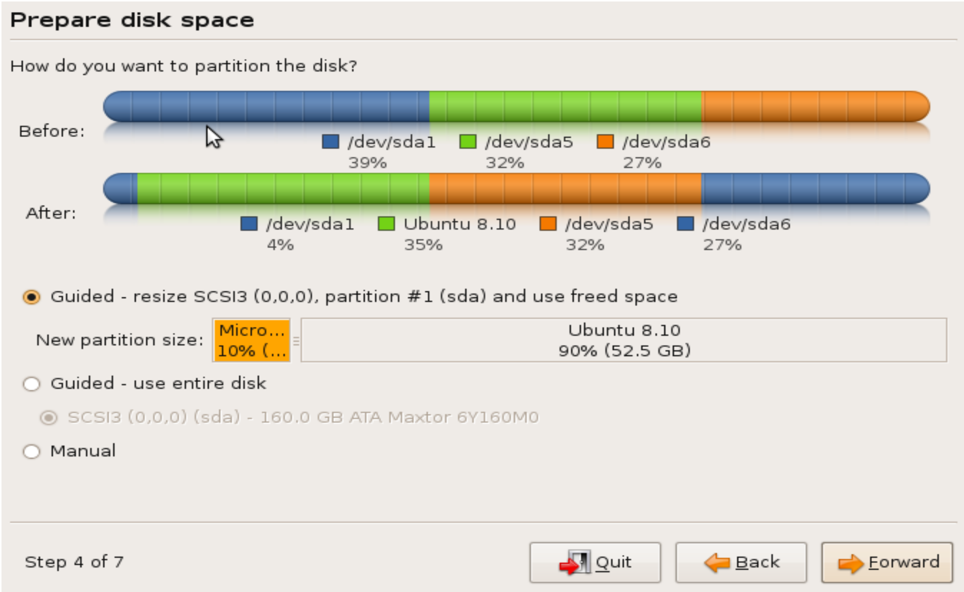
2. أختار منطقتك (أين أنت) مثلا (الرياض).



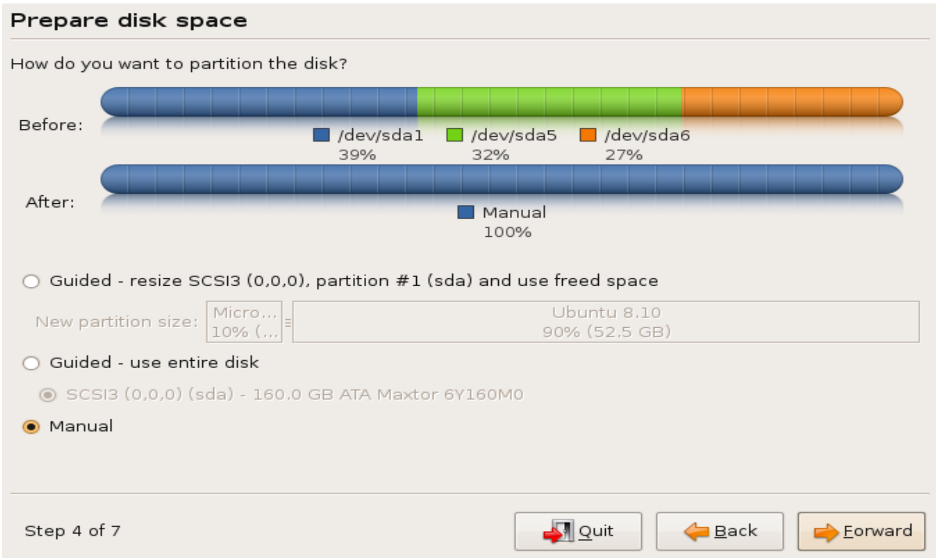
3. أختار لغة الكتابة التي تريد استخدامها بالإضافة إلى الانجليزية (لوحة المفاتيح)، مثلا Arabic -> Arabic-qwert/digits



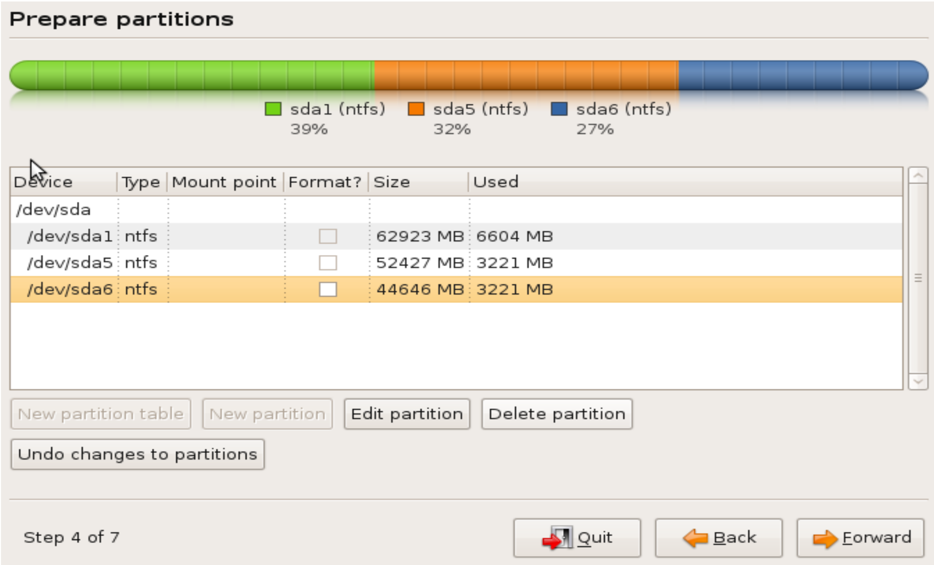
4. تهيئة جزء القرص الصلب الذي تريد تثبيت أوبونتو عليه:



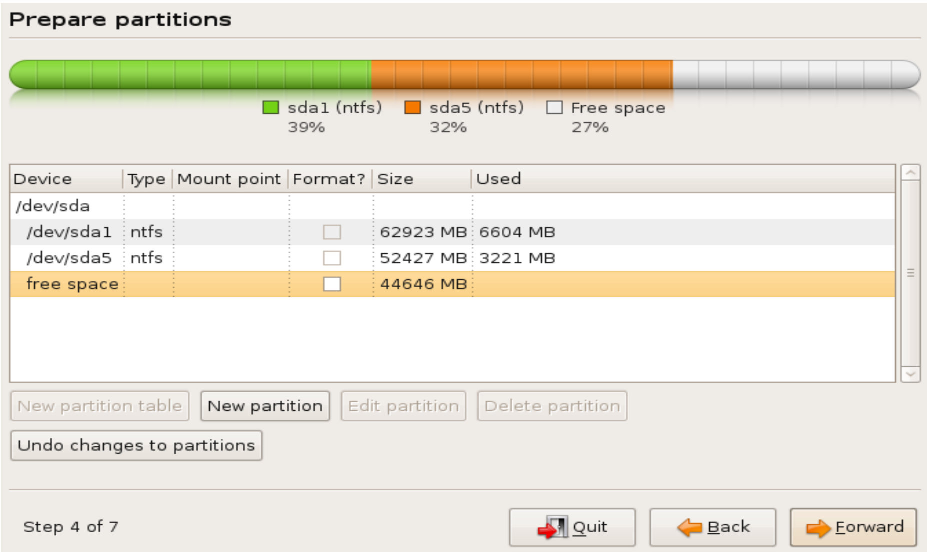
a. اختار manual



b. تظهر النافذة التالية:

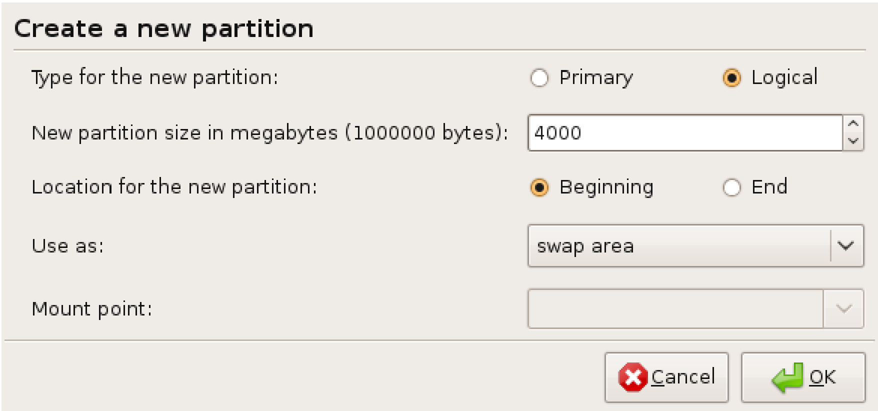


c. هنا القرص الصلب مقسم إلى ثلاث أقسام (هي c، D، E)، سأقوم بحذف القسم التالي E وذلك بالنقر عليه ، ثم أختيار Delete partition.

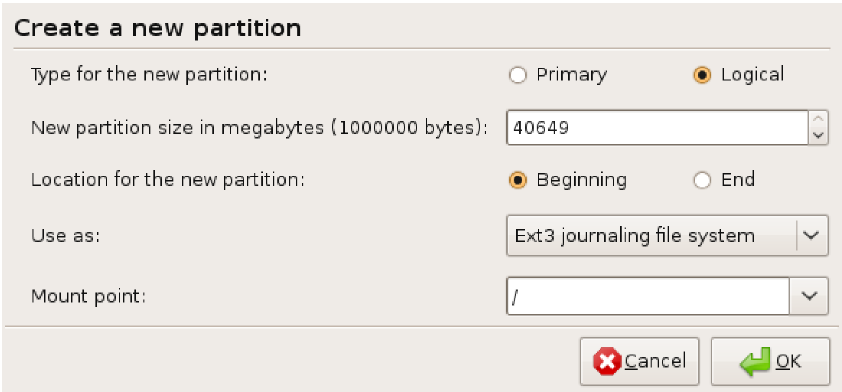


d. سيصبح الجزء غير مستخدم (free space)، انقر عليه ثم اختار إنشاء جديد

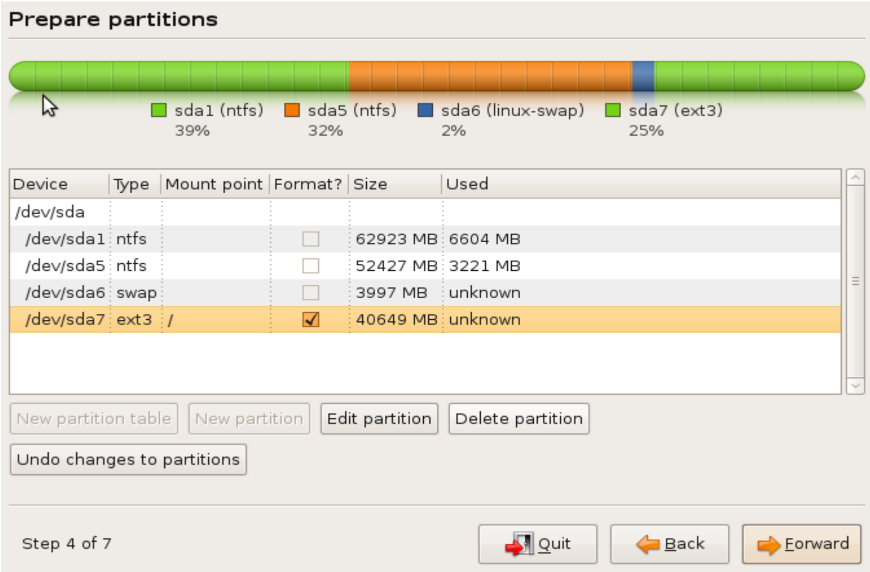
e. اختار في مربع المساحة قيمة من مضاعفات الذاكرة (الرام) مثلا 2000 أو 4000) وحدد نوع المساحة على أنها مساحة إبدال (swap) ، ثم انقر على موافق.



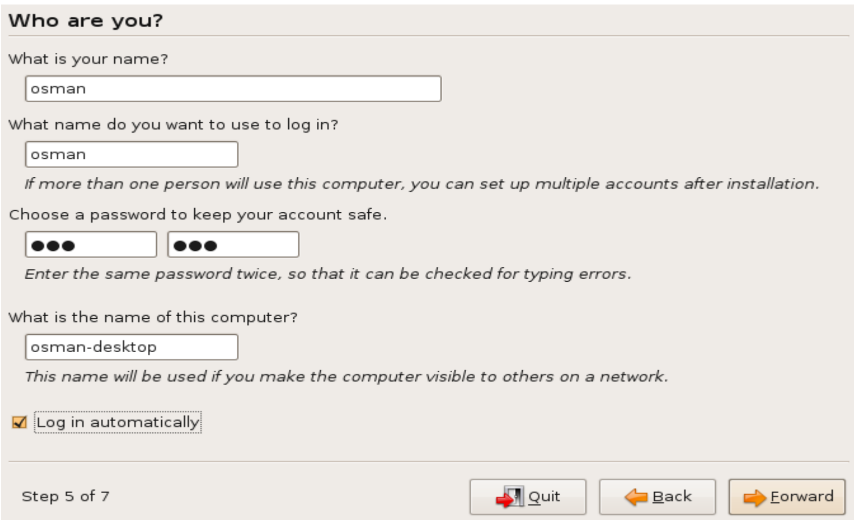
f. انقر على باقي الجزء غير المستخدم ثم إنشاء جديد، هنا اترك باقي المساحة كما هي (المساحة التي سنثبت فيها أوبونتو)، وضع في المربع الاخير الفارغ علامة "/" و هي علامة الجذر التي تعنى أن هذا الجزء سيكون مكان تثبيت أوبونتو، ثم أنقر على موافق.



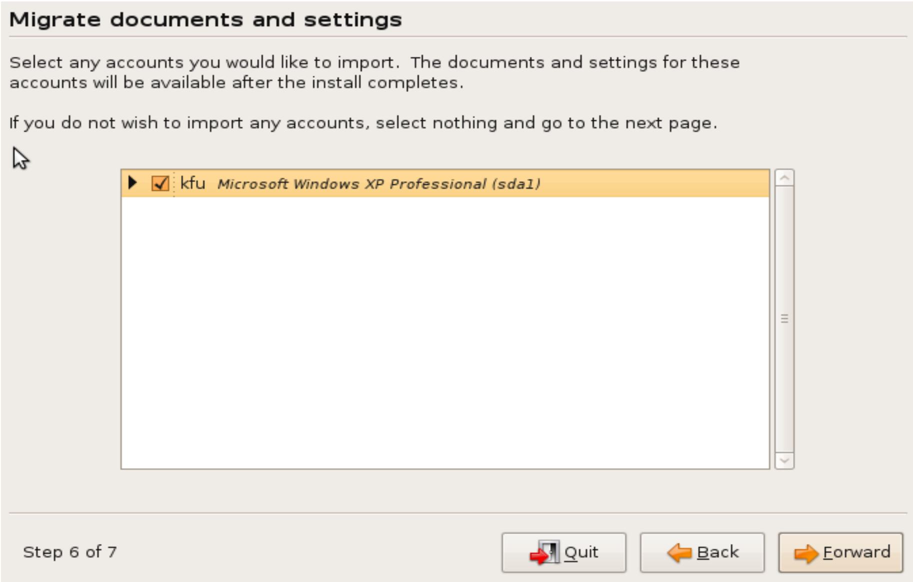
g. سيكون الشكل النهائي للتجزئية كما يلي:



5. ادخل بياناتك مثل اسم المستخدم وكلمة المرور واسم الحاسب.



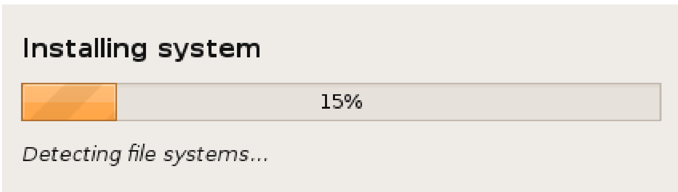
6. إذا أردت استيراد معلوماتك الموجودة في windows XP أنقر على خيار الاستيراد (عادة لانفعل ذلك).



7. أنقر على تثبيت.



8. فيبدأ التثبيت على جهازك حتى يكتمل.



9. بعد إكتمال التثبيت سيطلب منك النظام إعادة التشغيل، قم بذلك.

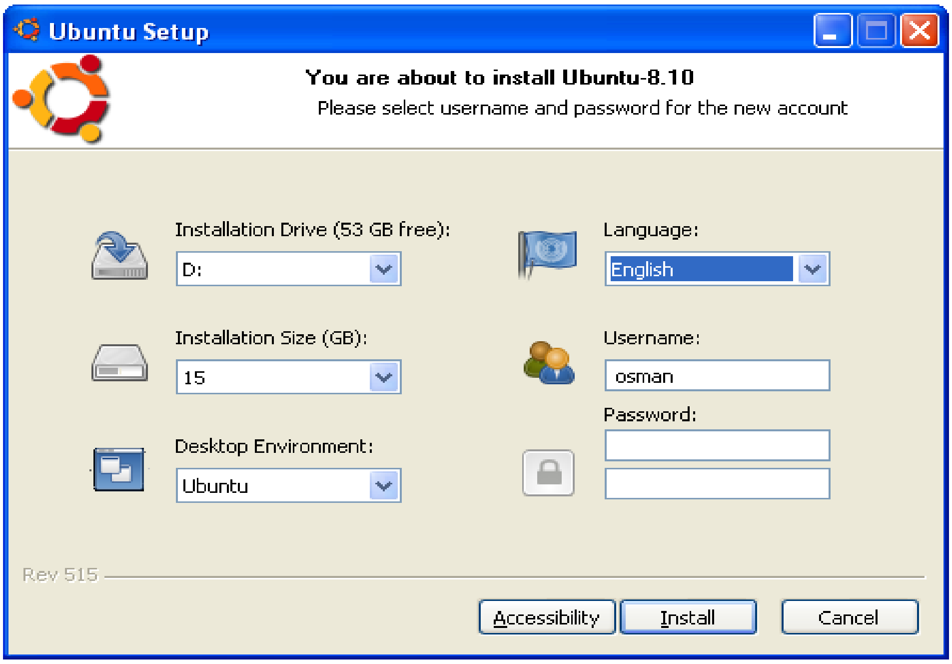
10. بعد إعادة التشغيل ستظهر عدة خيارات : الخيار الأول لتشغيل أوبونتو وهنالك خيار تشغيل نظام ويندوز السابق.

تنبيه هام: قبل تثبيت الأبونتو لابد من تقسيم المساحة الخالية المتروكة له إلى جزئيين هما:

- مساحة الابدال swap: وتكون من مضاعفات الذاكرة ، وهذه يستفيد منها النظام في وضع البيانات مؤقتا ولن نحتاجها بعد اتثبيت، وكلما زاد حجمها كلما زادت سرعة التنصيب.
- الجذر /: وهو المساحة التي يتم فيها تثبيت النظام وتستفيد من باقيها في تثبيت برامج إضافية على الابونتو، فإذا أردت استخدام الابونتو وتثبيت برامج كثيرة عليها فأحجز مساحة أكبر هنا.

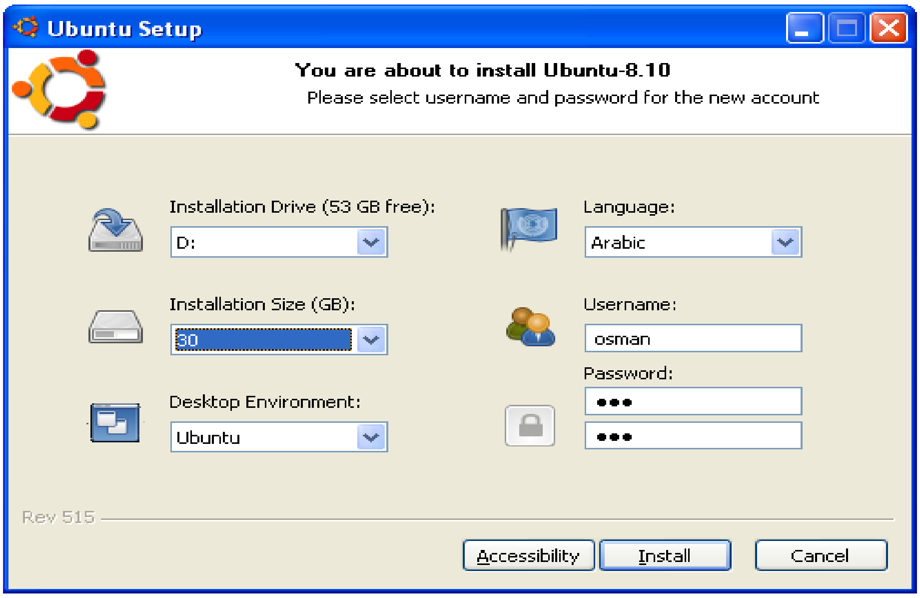
الخيار الثاني (التثبيت داخل ويندوز)

يمكنك تثبيت أوبونتو داخل ويندوز، وهنا لا تحتاج تقسيم القرص الصلب إلى أجزاء وإنما يكفي أن يكون لديك مساحة كافية للتثبيت (أقل شيء تكون 10GB)، فيتم تثبيت أوبونتو كملف عادي داخل ويندوز، وهذه تعتبر طريقة سهلة جدا لتثبيت أوبونتو، فما عليك سوى أختار Install inside windows ثم متابعة الخطوات التالية:

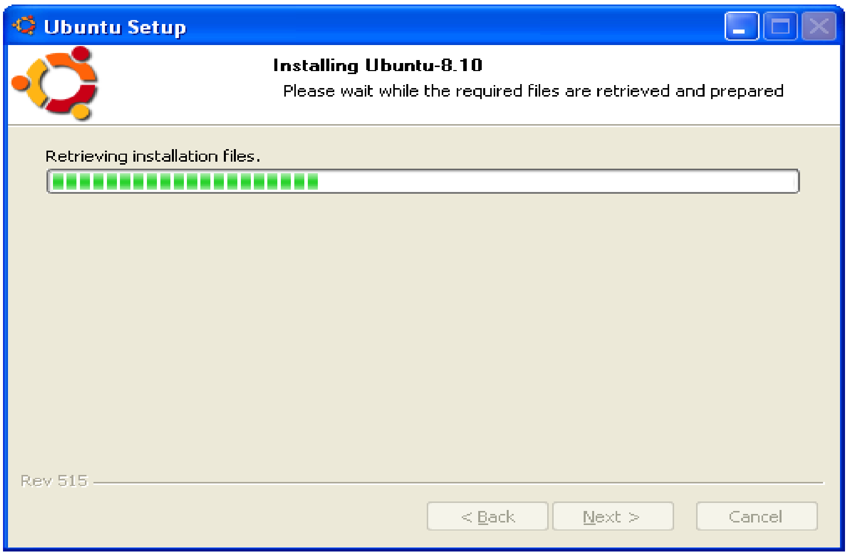


أختار لغة نظام التشغيل من (language)، ثم حدد مكان تثبيت أوبونتو في أي سواقة (مكان التثبيت) (installation drive)، ثم المساحة التي تريد حجزها لأوبونتو (installation size)، ثم اسم المستخدم (user name)، وكلمة المرور (password) التي نعيد كتابتها للتأكيد. بعدها أنقر على زر install.

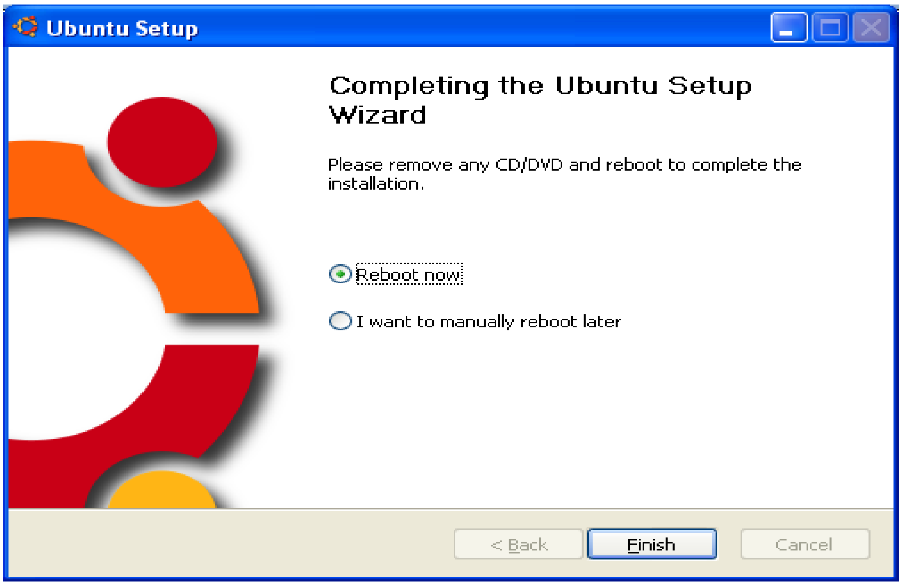
مثال لتعبئة الخيارات أعلاه :



يبدأ التثبيت:



بعد اكتمال التثبيت تظهر الشاشة التالية:



أختار Reboot now ثم أنقر على زر Finish، سيتم إعادة تشغيل الجهاز. عند ما يفتح الحاسب مرة أخرى تظهر خيارات (حسب نظام التشغيل السابق لديك) مثل:

- Windows XP operating System
- Ubuntu

إذا أردت أن تستخدم نظام التشغيل أوبونتو أضغط السهم إلى الأسفل لتكون على الخيار Ubuntu ثم أنقر Enter. إذا لم تختار أي خيار فسيتم تشغيل ويندوز تلقائيا.